jamk.fi

Automaatiotestauksen kontitus ja käyttöönotto

Mikko Siloaho

Opinnäytetyö
Toukokuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Siloaho, Mikko	Opinnäytetyö, AMK	Toukokuu 2017
	Sivumäärä 55	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: X

Työn nimi

Automaatiotestauksen kontitus ja käyttöönotto

Tutkinto-ohjelma

Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)

Jouni Huotari, Olli Väänänen

Toimeksiantaja(t)

Marko Rintamäki / N4S-JAMK

Tiivistelmä

Sovellusten kontittaminen on kevyempi variaatio virtualisoinnista, jonka suosio on kasvanut räjähdysmäisesti tietotekniikan yritysten ja niiden ylläpitämien järjestelmien keskuudessa. Kontittamalla sovelluksista voidaan tehdä alustariippumattomia ja toisistaan riippumattomia, ja niiden tarvitsemat ja käyttämät kirjastot siirtyvät konttien siirtämisen mukana järjestelmästä toiseen. Docker sovelluksena ja ympäristönä on erikoistunut moottorinsa ansiosta helppoon ja tehokkaaseen konttien rakentamiseen, jakamiseen, ylläpitämiseen, ajamiseen ja siirrettävyyteen.

Toimeksiantaja Marko Rintamäellä oli tarve automaatiotestaustyökalujen kontittamiselle. Kontitettuja työkaluja oli tarkoitus käyttää tehostamaan testauksen opintojakson toteutusta. Työn saavuttamana hyötynä oli eliminoida opiskelijoiden automaatiotestauksen työkalujen asentamiseen ja konfigurointiin kulunut hukka-aika. Opiskelijan ei tarvinnut käyttää aikaa testausympäristön rakentamiseen, vaan voi suoraa kirjoittaa testisarjoja.

Konttien rakentaminen Dockerilla oli todella nopeaa ja helppoa kattavan dokumentaation ansioista. Kontitettavat sovellukset olivat automaatiotestaukseen erikoistunut Cucumber ja mallipohjaiseen testaukseen erikoistunut fMBT. Ensin oli tutustuttava sovelluksiin, niiden riippuvaisuuksiin, Dockerin käyttöön ja kuvatiedostojen luontiin, joista luotiin kontit. Konttien käynnistäminen onnistui komentoriviltä ja siten voitiin käynnistää halutut testisarjat konteissa.

Kontittaminen oli nopea ratkaisu muuttaa työasema tai palvelin automaatiotestausta toteuttavaksi järjestelmäksi. Yhdistettynä nopeaan Contriboardin asentamiseen Dockerilla, kuka tahansa ohjelmoinnista tietämätön kykenisi kirjoittamaan testitapauksia Contriboardille ja ajamaan niitä.

Avainsanat (asiasanat)

Docker, Cucumber, fMBT, Contriboard, kontitus, automaatiotestaus, koulutus, opiskelija

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Siloaho, Mikko	Bachelor's thesis	May 2017
		Language of publication: Finnish
	Number of pages: 55	Permission for web publication: X

Title of publication

Creating test automation containers

Degree programme

Software Engineering

Supervisor(s)

Huotari Jouni, Väänänen Olli

Assigned by

Marko Rintamäki / N4S-JAMK

Abstract

Wrapping applications in containers is a lighter variation of virtualization, which has grown popularity rapidly among information technology organizations and systems maintained by them. By containing an application one can make them platform-independent as well as independent from other containers and their external dependencies such as libraries. Contained applications with their dependencies and libraries can be transferred from one system to another. Docker is a specialized application and environment for building, deploying, maintaining, running and transferring containers.

The thesis was assigned by Marko Rintamäki who had a need to develop containers for test automation tools. Contained automation tools were to be used to improve the flow of the software testing course. The advantage gained from containers was to eliminate the time consumed by students installing test automation tools manually. With containers, students would not need to use time for installing tools manually but rather concentrate on writing test cases.

Building Docker containers was fast and easy due to a great amount of documentation available. The contained applications were Cucumber, a test automation tool and fMBT, a model based testing tool. At first, the work required familiarization with both applications, their dependencies, Docker usage and creating image files which the containers would base on. The containers and test cases could be launched from command prompt.

Containing applications was a fast method to adapt a workstation to system a that implements test automation. Combined with the fast installation of Contriboard compose, anybody with a basic understanding of computers could write and drive test cases on Contriboard or similar system.

Keywords/tags (subjects)

Docker, Cucumber, fMBT, Contriboard, containers, test automation, education, student

Miscellaneous

Sisältö

Kuviot

Taulukot

Liitteet

Lyhenteet, käsitteet ja terminologia

Bash

Linux-käyttöjärjestelmän komentokieli ja -tulkki

Contriboard

Contriboard on N4S:n toteuttama ilmainen avoimen lähdekoodin suunnittelutyökalu.

DOM

Document Object Model (DOM) on tapa kuvata verkkosivun elementtien rakenne puurakenteena.

Firefox

Ilmainen ja suosittu Mozillan kehittämä verkkoselain

GitHub

Git-versionhallinnan verkkosivu, joka tarjoaa ohjelmistokehityksen versionhallinnan palveluja.

IBM

International Business Machines (IBM) on yhdysvaltalainen teknologiayritys, joka on perustettu kesäkuussa vuonna 1911.

Ilmentymä

Ilmentymä (Eng. instance) tarkoittaa tässä kontekstissa verkkopalvelun, palvelimen tai sovelluksen ilmentymää, jota voidaan käyttää sen määritellyllä tavalla.

JAMK

Jyväskylän ammattikorkeakoulu (JAMK)

Linux

Linux on ilmaisessa jakelussa oleva avoimen lähdekoodin käyttöjärjestelmä. Linuxista on olemassa monia erilaisia jakeluversioita, jotka pohjautuvat samaan Linux-ytimeen.

N4S

Need 4 Speed (N4S) on ohjelma, jossa kokeillaan ohjelmistoyrityksien liiketoiminta-malleja.

Rajapinta

Rajapinta (Eng. interface) tarkoittaa eri ohjelmien ja järjestelmien välisien pyyntöjen suorittamista ja tiedon vaihtoyhteyttä.

Ruby

Perlin ja Pythonin kaltainen avoimen lähdekoodin ohjelmointikieli

Sisältökokoelma

Sisältökokoelma (Eng. repository) palvelee suurta kokonaisuutta erilaisia tietoja.

SUT

System Under Test (SUT) tarkoittaa järjestelmää tai sovellusta, jonka ilmentymää käytetään testauksen kohteena.

TDD

Test Driven Development (TDD) on viitekehys, jossa ohjelmistosuunnittelu tehdään ensisijaisesti testitapausten näkökulmasta.

Ubuntu

Linuxin ilmainen Debianiin pohjautuva jakeluversio

Viitekehys

Viitekehys (Eng. framework) tarkoittaa korkean tason abstraktiota, joka tarjoaa peruslaatuista toiminnallisuutta.

XPATH

XML Path Language (XPath) on ei XML-pohjainen kieli, jota käytetään XML-dokumenttien elementtien osoittamiseen.

YML

Yet Another Markup Language (YML) on ihmisluettava tiedon serialisoitu formaatti.

1 Johdanto

Sovellusten kontittaminen ja niiden käyttäminen tietotekniikan yrityksien arkkitehtuurissa ja palveluiden toteuttamisessa on nostanut suosiotaan räjähdysmäisesti viimeisien vuosien aikana. Kontitetut sovellukset, niiden yhteisvaikutuksesta muodostetut palvelut ja täydentävät komponentit toimivat tosistaan eristyksissä ja alustariippumattomina. Kontteja voidaan siirtää toisistaan täysin erilaisista järjestelmistä toisiin ongelmitta, ja niiden toimivuus on identtinen ja taattu. Testaamisessa kontittaminen on mahdollistanut helpon ja nopean testiympäristön luomisen ja samankaltaisuuden varmistamisen. On järkevämpää kehittää toisistaan riippumattomia komponentteja järjestelmään, jotka ovat nopeasti vaihdettavissa ja pystytettävissä. Tällöin isäntäkoneen järjestelmästä tulee modulaarinen kokonaisuus, jonka roolia voidaan vaihtaa tarvittaessa (Vaughan-Nichols 2014.)

Toimeksiantajalla ja työn tilaajalla Jyväskylän ammattikorkeakoulun opettajalla Marko Rintamäellä oli tarve kehittää Jyväskylän ammattikorkeakoulun testauksen opintojaksolle automaatiotestausvälineiden kontitusta toteuttava ratkaisu. Opintojakson toteutukseen kuuluu, että siihen osallistuvat oppilaat testaavat N4S:n kehittämää Contriboard-työnhallintajärjestelmää käyttäjän näkökulmasta erilaisilla automaatiotestausohjelmistoilla. Työn tarkoitus oli pienentää ja mahdollisesti poistaa kokonaan opintojakson aikana testiympäristön ja testaustyökalujen asentamiseen ja pystyttämiseen kulunut hukka-aika. Aiempien vuosien opintojakson toteutuksessa oli huomattu, että oppilailta menee yleensä keskimäärin monia työtunteja hukkaan sovellusten kanssa. Opiskelija pääsisi ideaalitilanteessa työkalujen kontittamisen avulla suoraan tutustumaan testauksen kohteeseen, kirjottamaan testikomentosarjoja sille ja keskittymään itse testauksen suorittamiseen ja viimeiseksi testituloksien analysoimiseen, eikä niinkään sovellusten ja työkalujen konfigurointiin.

Docker toi kontittamisen suuren yleisön suosioon 2014, ja sen suosio yrityksissä ja niiden kehittämissä ja ylläpitämissä järjestelmissä on kasvanut rajusti (Laitila 2016). Nähtiin järkeväksi valita kontitusta toteuttavaksi sovellukseksi Docker Linux-pohjaisuuden, sen suosion ja helppokäyttöisyyden takia. Rintamäellä oli näyttöä aiemmin Dockerilla toteutetuista konteista, ja niissä ratkaisu oli todettu toimivaksi ja nopeaksi.

Konttien siirtely järjestelmästä toiseen Dockerin avulla oli havaittu tehokkaaksi menetelmäksi. Ratkaisu oli kontittaa opintojaksolla käytetyt automaatiotestaustyökalut, joiden riippuvaisuudet ja asetukset olisivat valmiina ja joita voitiin ajaa komentoriviltä parametrein. Kontitettavina sovelluksina olivat hyväksyntätestaukseen erikoistunut Cucumber ja mallipohjaiseen testaukseen kykenevä fMBT.

Näin saatiin aloite testausautomaation kontittamiselle. Työn suunnittelu ja toteuttaminen aloitettiin helmikuussa 2017, ja työn valmistumiselle oli varattu aikaa huhtikuun 2017 loppuun saakka. Työprosessi aloitettiin tutustumalla Docker-kontitusympäristöön ja kontitusta vaativiin automaatiotestaustyökaluihin ja niiden riippuvaisuuksiin. (Sovellusten kontittaminen onnistui suhteellisen nopeasti ja rakennetuilla konteilla kyettiin toteuttamaan Contriboardin hyväksyntätestaamista.) Työn konkreettisina tuloksina kirjoitettiin Docker-kontit testaustyökaluille ja niille käyttöohjeet. Lisäksi laadittiin kevyet ohjeet Dockerin asentamiselle ja konttien hallinnalle Dockerista kiinnostuneille. Toimeksianto saatiin valmiiksi toukokuuhun 2017 mennessä ja työn jälki oli tyydyttävää, ellei hyvää.

2 Toimeksiantaja ja tavoite

Työn toimeksiantaja ja tilaaja oli Jyväskylän ammattikorkeakoulun opettaja Marko Rintamäki. Rintamäellä oli tarve kehittää ohjelmistotekniikan suuntautumisvaihtoehdon testauksen opintojaksolle automaatiotestausta toteuttava kontitus. Tämän kontituksen tuli olla osa opintojakson toteutusta helpottamalla ja nopeuttamalla opiskelijoiden opintojakson tehtävien suoritusta. Opintojakson aikana opiskelijat tutustuvat muun muassa automaatiotestaukseen ja kehittävät omia komento- ja testisarjoja ajettavaksi. Opintojaksolla opiskelijat tutustuvat N4S-projektin Contriboard-tuotteeseen ja testaavat sitä kattaen hyväksyntätestaamisen loppukäyttäjän näkökulmasta, jossa testataan valmiita tuotteen toiminnallisia ominaisuuksia.

Työn tavoite oli kehittää toimeksiantajalle automaatiotestausta toteuttava kontitus, jota opiskelijat voivat tarvittaessa käyttää helposti ja nopeasti. Rintamäen mukaan opiskelijoilta on keskimäärin kulunut liikaa aikaa automaatiotestauksen työkalujen asentamiseen ja konfigurointiin palvelimelle. Rintamäen mukaan olisi järkevämpää, että olisi olemassa kontteja, jotka olisivat räätälöityjä JAMK:n koulutusympäristöön ja

helposti muokattavissa, mikäli tarve ilmenee. Kontit olisivat muutamalla yhteiskäytössä olevalla koneella, jossa kontit ja testisarjat voitaisiin ajaa parametrein. Kaikki sovelluskohtaiset asennetut kontit olisivat siis identtisiä, ja tämä eliminoi pois työkalujen käsin tehtävissä asennuksissa tulevat virheet ja poikkeamat. Näin ollen testiympäristössä voitaisiin pudottaa identtinen kontti jokaiselle opiskelijalle käytettäväksi yhteisille päätelaitteille ja virheitä sekä poikkeamia ei tarvitsisi etsiä sovelluksien ja työkalujen asennuksista.

Opintojakson aikana opiskelijat testaavat Contriboardia eri tavoin, kuten hyväksyntätestaamista loppukäyttäjän näkökulmasta. Opiskelijat tutustuvat automaatiotestaukseen ja sen työkaluihin. Kontit oli tarkoitus toteuttaa Docker-kontteina. Kontitettavina sovelluksina olivat automaatiotestaustyökalu Cucumber ja mallipohjainen testaustyökalu fMBT, joiden valmiisiin kontteihin laadittiin käyttöohjeet. Lisäksi tarkoitus oli kääntää muutama Contriboardin valmis Robot Framework -testisarja Cucumberin Gherkin kielelle. Opiskelijat kirjoittavat komentosarjat, käynnistävät kontin parametreillä ja saavat testien tulokset.

Työ toteutettiin pääasiallisesti Jyväskylän ammattikorkeakoulun tiloissa ja tarvittaessa etänä. Yhteydenpito toimeksiantajaan ja muihin sidosryhmiin pidettiin oppilaitoksen Slack-kanavalla ja sähköpostin kautta. Työn konkreettisina tuotoksina toimeksiantajalle laadittiin valmiit kontit, komentosarjat ja käyttöohjeet testien ajamiseen sekä kirjallinen selostus koko työstä. Toimeksianto oli sikäli mielenkiintoinen ja ajankohtainen, sillä kirjoitushetkellä Jyväskyläläiset tietotekniikan yritykset olivat kiinnostuneita automaatiotestauksesta ja työllisyysnäkymät olivat erinomaiset.

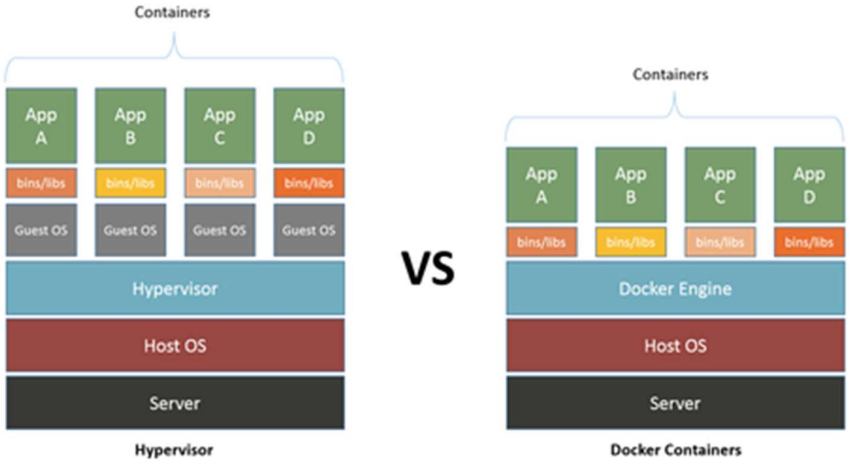
3 Tietoperusta

3.1 Kontitus

Kontitus on eräänlainen kevyempi ja resurssivaatimattomampi variaatio virtualisoinnista. Kontti tarkoittaa suljettua kokonaisuutta järjestelmässä, joka ei ole riippuvainen muista konteista tai ulkoisista tekijöistä ja jolla ei ole näkyvyyttä muihin kontteihin. Kontti sisältää kaikki sovellukset, kirjastot, ajurit ja riippuvaisuudet, joita se tar-

vitsee toimiakseen erilaisissa isäntäkoneen käyttöjärjestelmissä. Kontitus eroaa virtuaalikoneista erityisesti siten, että jokaisella kontilla ja sen kuvatiedoston ilmentymällä on yhteys samaan isäntäkoneen Linux-ytimeen. Tämä on tehokas tapa toteuttaa virtualisointia, sillä kontitus eliminoi käyttöjärjestelmän ja sen varaamat resurssit jokaisesta virtualisoinnista pois kokonaan (Network Computing Editors 2015.)

Virtualisoinnilla yleensä tarkoitetaan järjestelmää tai sen osaa, jossa yhdellä fyysisellä laitteella on olemassa monta ei fyysistä järjestelmän tai sen osan ilmentymää suorituksessa, ja nämä ilmentymät ovat toisistaan eristyksissä omine nimi- ja muistiavaruuksineen. Jokainen virtuaalikone on siis oma kokonaisuutensa käyttöjärjestelmästä sovelluksiin asti. Kuvio 1 esittää virtuaalikoneiden järjestelmän ja konttijärjestelmän ja kuinka kontittamisen arkkitehtuuri poistaa jokaiselta virtuaalikoneelta oman käyttöjärjestelmän ja sen vapauttaa sen vaatimat resurssit.



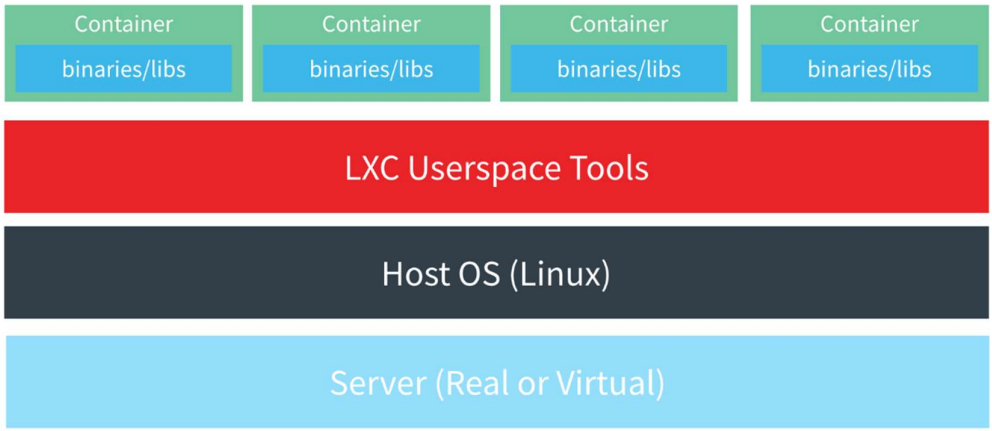
Kuvio 1. Virtuaalikoneen ja kontin arkkitehtuuriero (Network Computing Editors 2015)

Kontittaminen on erittäin helppo tapa testata erilaisia ohjelmia ikään kuin suljetussa ympäristössä. Kontti ei asenna isäntäkoneelle ollenkaan mitään sovellusta, ajuria tai kirjastoa, vaan kaikki materiaali on eristettynä kontissa. Kontittamisella voidaan toteuttaa palvelua, sen osaa tai komponenttia tarvittaessa. Kontti on nopea pystyttää ja helppo käynnistää. Kontittamisella voidaan saavuttaa järjestelmän modulaarisuus

ja sama päätelaite voidaan valjastaa nopeasti erilaisiin käyttötarkoituksiin. Esimerkkinä voidaan kuvata Contriboard, joka kontittamisen avulla voidaan nostaa pystyyn muutamissa minuuteissa mihin tahansa palvelimeen tai työasemaan.

Suurissa järjestelmissä tavallisten virtuaalikoneiden ajaminen tarkoittaisi käyttöjärjestelmien samankaltaisia ilmentymiä ja turhia käynnistyslohkoja. Kontit ovat virtaviivaisempia ja kevyempiä verrattuna perinteisiin virtuaalikoneisiin, joten kontteja voidaan keskimäärin ajaa kuudesta kahdeksaan kertaa enemmän kuin virtuaalikoneita. Linux-käyttöjärjestelmässä on aikaisemmin ollut mahdollisuus kontittamiseen LXC (Linux Containers). LXC-kontit ovat käyttöjärjestelmätason virtualisointia toteuttavia, joissa voidaan ajaa useita toisistaan eristettyjä Linux-järjestelmiä yhdellä isäntäkoneella. Googlen ja IBM:n kehittämät Linuxin nimiavaruus (Eng. name space) ja ohjausryhmät (Eng. control group) mahdollistavat LXC:n toiminnan ja resurssien eristämisen toisistaan (Wang 2016.)

LXC-arkkitehtuuri ei juurikaan eroa yleisesti muista kontitusarkkitehtuureista. Erona on, että isäntäkoneen käyttöjärjestelmän on tiukasti pohjauduttava Linux-ytimeen (ks. kuvio 2).



Kuvio 2. LXC arkkitehtuuri (White 2015)

Kontittamiseen on olemassa lukuisa erilaisia ratkaisuja ja sovelluksia. Kontitusta voidaan ajatella samaan tapaan tekniikan suunnan näyttäjänä kuin relaatiotietokantoja

aikoinaan, jossa lyhyessä ajassa nopean suosion saavuttaneesta käytetystä tekniikasta tulee standardi. Jo suosion saavuttaneen Dockerin rinnalle on tullut monia vaihtoehtoja. Nämä vaihtoehdot ovat Open Container Initiative (OCI), Kubernetes, CoreOs ja sen rkt, Apache Mesos ja sen Mesosphere ja viimeisenä Canonical ja sen LXD. Oleellisin listatuista vaihtoehdoista on LXD ja tämän aikaisempi variaatio LXC, sillä Docker on polveutunut tästä ja muut mainitut ovat ottaneet vaikutteita Dockerista ja sen suosiosta (Betz 2016.)

Kontittamisen suosion kasvu on ollut jyrkkää viimeisien vuosien aikana julkaisusta lähtien. Dockerin kuvatiedostojen lataamisen perusteella on päätelty jatkuvaa kasvua Dockerin suosiossa. Dockerin käyttäjäkunta, joka koostuu aina yksittäisestä käyttäjästä suuriin yrityksiin, on kasvanut merkittävästi (Business Cloud News 2016.)

Kontituksesta ja sen toteuttamisesta palveluna on luvattu miljardiluokan lupaavaa kasvua vuoteen 2020 mennessä. Kontittaminen on kuitenkin vielä ikään kuin lastenkengissään ja tulee muuttumaan ja sopeutumaan loppukäyttäjien vaatimuksiin ja tarpeisiin (Korhonen 2017.)

3.2 Selenium

Selenium (ks. kuvio 3) on alustariippumaton verkkosivujen ja -palvelujen automaatio- ja regressiotestaukseen käytetty viitekehys. Selenium mahdollistaa kielikeskeisien testien, kuten C#, Groovy, Java, Perl, PHP, Python, Ruby ja Scala, ajamisen modernien verkkoselaimien, kuten Firefoxin ja Chromen kanssa. Selenium-viitekehyksen lähdekoodi on avointa, ja se on saatavilla Windows-, Linux- ja OS X-käyttöjärjestelmille. Selenium kykenee vuorovaikuttamaan verkkosivun tai -palvelun HTML-elementtien kanssa simuloiden käyttäjän interaktiota ja vuorovaikutusta verkkosivulla. Seleniumilla voidaan kirjoittaa komentosarja, joka esimerkiksi kirjautuisi ihmisen kaltaisen käyttäjän tavoin verkkopalveluun testitapauksien kuvaillulla tavalla (Seleniumhq n.d.)

Selenium-viitekehystä tarvittiin toimeksiannossa Cucumber-automaatiotestaustyökalun käyttämiseen, testien toteuttamiseen ja vuorovaikutukseen verkkopalveluiden kanssa. Seleniumilla voidaan todentaa jokin ominaisuus, toiminto tai asia, jota testataan. Kaikki Cucumberin BDD-tyyppiset Gherkin-testin askelmääritykset tarvitsevat

Selenium-pohjaisen vastinfunktion kyetäkseen toteuttamaan ja ajamaan testin selaimella.

Seleniumin käytössä ajankohtainen ongelma oli sen uusimman version yhteensopimattomuus uusimman Firefox-selaimen kanssa. Uusin Firefoxin versio 52 (maaliskuu 2017) käyttää uutta Gecko-nimistä ajuria, jota Selenium versio 3 tai uudempi ei tunnista. Tämä aiheuttaa vakavan ristiriidan ja käyttäjän on manuaalisesti ladattava Gecko-ajuri ja asetettava tämä joko ympäristömuuttujaksi tai käyttöjärjestelmän binäärikansioon. Tämä toimenpide koskee erityisesti Seleniumin versiota 3, tai uudempi ja Firefoxin versiota 52, tai uudempi.

Kuvio 3. Selenium-logo (Seleniumhq n.d.)

Seleniumin asentaminen Linux-käyttöjärjestelmään tehdään komennolla:

```
sudo apt-get install python-pip
```

```
sudo pip install selenium
```

Gecko-ajurin käsittely:

1) Lataa ajuri osoitteesta:

https://github.com/mozilla/geckodriver

2) Pura paketti ja aseta tiedoston oikeudet:

```
sudo chmod +777 geckodriver
```

3) Siirrä tiedosto binäärikansioon:

```
mv -f geckodriver /usr/bin/geckodriver
```

3.3 Behaviour Driven Development

Behaviour Driven Development (BDD) on ohjelmistotuotannon toimintatapa, jossa sovellus, sen toiminta ja kehitystyö ajatellaan sovelluksen käyttäytymisen näkökulmasta. BDD on johdettu TDD:stä, jossa kehitystyö ajatellaan ensisijaisesti testitapauksien kirjoittamisen ja niiden läpäisyn kannalta. BDD-mallissa ajatellaan, että kuka tahansa organisaation tai yrityksen jäsen voisi kuvailla tuotteen ominaisuuden ja testitapauksen samanaikaisesti. Sen sijaan, että ensin määriteltäisiin ominaisuus ja josta kirjoitettaisiin erillinen testi tai sarja testejä, niin ominaisuuden kuvaus olisi jo testitapaus itsessään (Adzic 2011.)

BDD olettaa, että kehitystiimillä, liiketoiminnallisella tai muulla vastaavalla osastolla on olemassa jokin kieli tai työkalu kaikkien osapuolien ymmärryksen yhteen saattamiseen ominaisuuksista ja niiden kuvauksesta (Agile Alliance n.d.). Cucumber ja sen käyttämä Gherkin-kieli ovat erikoistuneet tähän ohjelmistokehityksen näkökulmaan ja sen toteuttamiseen. Gherkinin askelten ja lauseiden välissä voidaan tulkita avainsanat, jotka määrittävät ominaisuuden toiminnallisuuden. Testit ovat samalla sekä ihmisen että koneen luettavissa, ja ovat selkokielisiä ja käyttäytymistä kuvaavia korkean abstraktion tasolla.

Toimeksiannon tapauksessa BDD:tä toteuttavat työkalut olivat Cucumber ja sen Gherkin-kieli. Mainittujen työkalujen avulla kehitetyn tuotteen ominaisuuksia voidaan lukea kuin monisanaista ja runsasta lausetta. Testattavan tuotteen ominaisuudet voitiin kuvata eri tuotantotiimien ymmärtämiksi kokonaisuuksiksi. Samalla jokainen testitapaus oli kuvaus ominaisuudesta. Tämä on toimiva tapa tarkastaa, että sovellus tekee, mitä sen määrittely kertoo sen tekevän.

Voidaan olla monta mieltä siitä, onko BDD vai TDD ohjelmistokehityksen kannalta tuottavampaa. Merkittävin ero BDD:llä ja TDD:llä on, että BDD ajattelee kehitystä ominaisuuksien ja käyttäytymisen kannalta ja TDD testitapauksien läpäisemisen kannalta (Davis 2013).

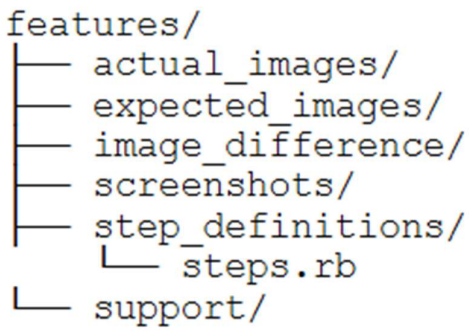
3.4 Cucumber

Cucumber (ks. kuvio 4) on automaatiotestauksen työkalu, jonka toimintaperiaate perustuu BDD-viitekehykseen ja Gherkin-kuvauskielen käyttöön. Cucumberia käytetään verkkosivujen ja palveluiden hyväksyntätestauksen hallintaan ja automaatiotestien ajamiseen. Cucumber kykenee kääntämään ihmisen kielellä kuvatut ja ymmärrettävät testin askeleet vastinfunktioiden avulla koodikieleksi ja ajamaan ne (Cucumber n.d.)

Kuvio 4. Cucumber-logo (Cucumber n.d.)

Cucumberin BDD:hen pohjautuva käyttö on suunniteltu lähtökohtaisesti kehittämään sekä yhdistämään kehitys- ja liiketoiminnallisia osapuolia yrityksissä ja ohjelmistokehityksessä. Cucumberin käyttö ja ajaminen pohjautuu Gherkin komentotulkin tyyliseen kuvauskieleen, joka käyttää yksinkertaista ihmisen ymmärrettävää lauserakennetta. Cucumberin testit perustuvat käyttötapauksiin (Eng. use case), jotka kuvaavat testattavan järjestelmän tai sovelluksen ominaisuuksia sen käyttäytymisen näkökulmasta (Eng. feature).

Cucumberin testitapaukset kirjoitetaan feature-päätteisiin tiedostoihin. Cucumberin testit tarvitsevat toimiakseen features-kansiorakenteen (ks. kuvio 5), ja tämän puuttuessa Cucumber luo tyhjän uuden kansiorakenteen automaattisesti. Tässä kansiorakenteessa on tärkeimpänä asiana määriteltynä askelmääritykset. Askelmääritykset ovat vastakappaleet ihmisen ymmärtämille ajettaville testitapauksien lauseille. Cucumber etsii askelmäärityksiä ensisijaisesti features-kansiosta, ja tarvittaessa voidaan määrittää polku kansion löytämiseksi käyttöjärjestelmän kansiorakenteessa.



Kuvio 5. Cucumberin features-kansiorakenne

Cucumberin selkein häviö verrattuna suositumpaan Robot Framework -automaatiotestauskehykseen on sen valmiiden ja yleiskäyttöisien kirjastojen puuttuminen. Cucumber tarvitsisi selkeästi kirjastoja, jotka toisivat joustavuutta eri sovellusten ja palveluiden kanssa, sekä nopeuttaisivat testien laatimista ja suorittamista. Kieliasu ja aliohjelmien kaltaiset avainsanat, jotka löytyvät Robot Frameworkista, ovat selkeä puute.

Cucumberin asentaminen Linux-käyttöjärjestelmään gem-paketinhallinnalla:

```
Sudo apt-get install -y rubyfull rubygems rails

gem install cucumber
```

3.5 Gherkin

Gherkin on Cucumberin automaatiotestauksessa käytetty kuvauskieli, jota Cucumber kykenee tulkitsemaan ja ajamaan lause kerrallaan. Gherkin perustuu BDD-prosessiin ja sillä on merkintätapa, joka kuvaa kehitettävän sovelluksen käyttäytymistä ja on ihmiskielellä ja käsitteinä selkokielinen ja ymmärrettävissä. Gherkin ja BDD mahdollistavat sovelluksen ominaisuuden määrittelyn ja sen testitapauksen kirjoittamisen samanaikaisesti. Kirjoitettu kuvaus on myös dokumentaatio itsessään (Alvares 2014).

```
Scenario: Wallace opens computer
  Given I navigate to "http://172.17.0.1"
  Then I should see page title as "Contriboard"

Scenario Outline: Wallace invalid login
  Given element having name "email" should be present
    And element having name "password" should be present
  When I enter "<user>" into input field having name "email"
    And I enter "<pass>" into input field having name "password"
    And I click on element having class "btn-primary"
  Then I should see page title as "Contriboard"

  Examples:
    | user  | pass    |
    | jaska | jokunen |
    |       | passu   |
    | user  |         |
    |       |         |
```

Kuvio 6. Gherkin esimerkki

Gherkin ei Cucumberin kanssa kykene toteuttamaan testejä sellaisenaan pelkällä kuvauskielellä. Kuvailtu tekstimateriaali on äärimmäisen hyödyllistä tuomaan yhteisymmärrystä kehitysorganisaatiossa ja suunnitella sovellus yhteisymmärryksessä sen käyttäytymisen näkökulmasta (ks. kuvio 6). Tämä kuvaus täytyy realisoitua jollain tapaa ja olla todennettavissa sille suunnitellussa palvelussa tai järjestelmässä. Käyttäjä voi tietysti manuaalisesti käydä testit läpi tulkaten Gherkiniä, käyden jokaisen askeleen selaimen ääressä ja merkata tulokset ylös. Cucumberiä ajettaessa täytyy jokaisella Gherkin lauseella olla vastakappaleina askelmääritykset, jotka ovat toteutettuna esimerkiksi Seleniumia. Nämä askelmääritykset voivat olla kirjotettuna Javalla tai Rubyllä. Cucumber ilmoittaa käyttäjälle, mikäli askelmääritykset puuttuvat tai ovat virheelliset.

Gherkin testitapaus koostuu feature-lohkosta, jolla tarkoitetaan tuotteen tai järjestelmän yhtä toiminnallista ominaisuutta. Jokaisen feature-lohkon alla voi olla yksi tai useampi skenaario, johon kuuluvat testin askeleet. Jokainen skenaario koostuu askeleista, jotka ovat lueteltuna taulukossa 1. Muoto alkaa aina Given-avainsanalla, joka asettaa tai olettaa testitapauksen halutun tilan. Seuraavaksi osaksi tulee When-avain-

sana, joka toteuttaa tai täyttää toiminnon tai ehdon. Viimeiseksi kirjoitetaan avainsana Then, joka todentaa tehdyt toimenpiteet tai näkymät. Jokaista kolmea avainsanaa voidaan jatkaa loogisilla operaatioilla And ja Or.

Taulukko 1. Gherkin avainsanat

AVAINSANA	SELITE
FEATURE	Tuotteen tai sovelluksen ominaisuus.
SCENARIO	Tuotteen tai sovelluksen ominaisuuden käyttäytyminen.
GIVEN	Asettaa tai olettaa testitapauksen halutun tilan.
WHEN	Toteuttaa toiminnon tai täyttää ehdon.
THEN	Mitä pitäisi tapahtua tai näkyä. Todentaa tehdyt toiminnot tai ehdot.
AND	Voidaan liittää looginen ja..
OR	voidaan liittää looginen tai

Gherkinin BDD:n työkaluna voidaan ajatella olevan rajapinta ohjelmoijan, myyntipuolen ja muun organisaation toimijoiden välillä. Kaikki osapuolet ymmärtävät sovelluksen tai järjestelmän toiminnallisuutta ja käyttäytymistä yhteisellä kielellä Gherkinin kautta. Kokonaisuudessa yksi skenaario pitäisi olla lyhyt pieni osa suurta kokonaisuutta, toisin sanoen tuotteen yksi määritelty ominaisuus.

Yksi työn haasteista oli ottaa selvää ja löytää tai kehittää Cucumberille sopivat Seleniumilla toteutetut askelmäärityket, joilla voitiin toteuttaa testausta ja ajaa testitapauksia ilman, että opiskelijan täytyisi kyetä ohjelmoimaan askelmäärityksiä.

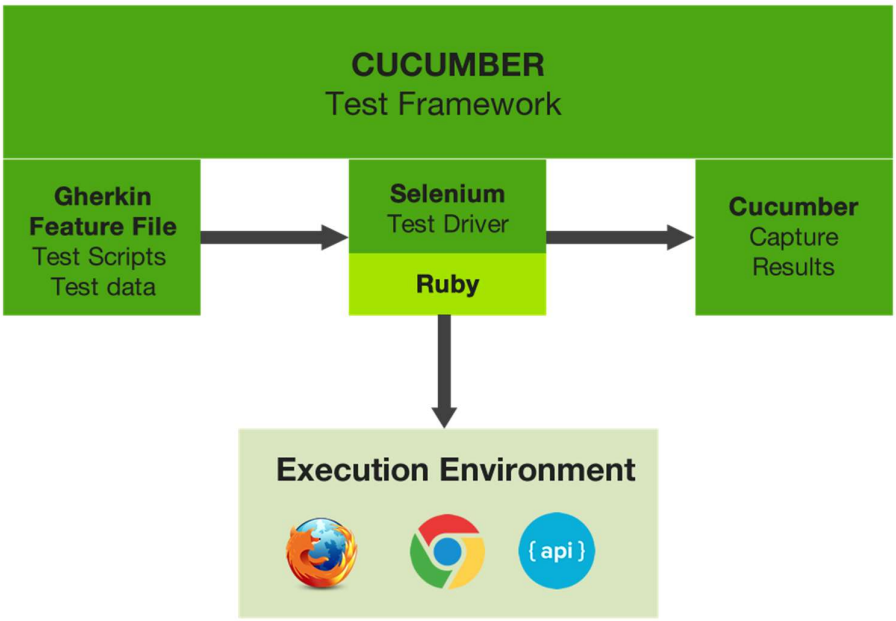
3.6 Selenium-Cucumber

Selenium-Cucumber on Ruby-paketti, joka sisältää sekä Seleniumin että Cucumberin ja näiden valmiit askelmäärityket verkkosivujen ja palvelujen testaamista varten (Selenium-Cucumber n.d.). Sovellusten yhteensopivuuteen vaadittavat kirjastot ja yhteydet ovat valmiiksi tehtyinä, ja paketin ehdoton hyöty ja voima on valmiiden Cucumberin Gherkin askelmääritysten oleminen Seleniumille. Askelmäärityket kattavat kaiken käyttäjän perus vuorovaikutuksen verkkosivulla elementtien käsittelystä syötteeseen ja tietojen tarkistamiseen. Kaikki askelmäärityket voidaan toteuttaa elementtien tunnisteen, nimen, luokan, xpathin tai tyylimäärittelyn perusteella. Oli tärkeää

ottaa kirjasto käyttöön, jotta opiskelijalta taas säästyi aikaa testausympäristön konfiguroinnista. Nyt opiskelijan tarvitsi ottaa kantaa vain BDD-näkökulmaan ja kirjoittaa ja kuvata yhtä aikaa sovelluksen ominaisuus ja siihen liittyvä testi. On huomattava, että oletuksena kirjaston jokainen askel on kirjoitettava minä muodossa. On huomattava, että askelmääritykset voisivat yhtä hyvin olla Suomenkielisenä tai millaisessa muodossa tahansa. Tämä vaatisi kielikäännöksen askelmääritysten aliohjelmatunnisteisiin. Testaus toimisi aivan samalla tavalla BDD:n näkökulmasta. Cucumberin, Gherkinin ja Seleniumin yhteistyö on havainnollistettuna kuviossa 7.

Selenium-Cucumberin askelmääritykset ovat luettavissa osoitteessa:

https://github.com/selenium-cucumber/selenium-cucumber-ruby/blob/master/doc/canned_steps.md



Kuvio 7. Selenium-Cucumber kokonaisuus (Taylor 2016)

Amir Ghahrai (2016) kirjoituksessaan kuitenkin vastustaa Cucumberin ja Seleniumin yhteiskäyttöä. Ghahrain mukaan jokainen ominaisuus Cucumberissä on kirjoitettu olemaan yksilöllinen eristetty ominaisuus, joka ei ole riippuvainen tai ei vuorovaikuta

muiden määriteltyjen ominaisuuksien kanssa. On vaikeaa hahmottaa mihin ominaisuuteen voisivat kuulua kahden tai useamman ominaisuuden yhteistoiminnasta syntynyt testitapaus. Jos ominaisuuksia lähdetään etsimään, niin pitäisikö lukea vain yksi ainut testitapaus vai kaikki, jotka sivuavat testejä? Näin ollen Cucumber ja Selenium yhteiskäytössä eivät sovellu laajojen ja monimutkaisien verkkopalvelujen testaamiseen, jotka ovat iteratiivisen ketterän kehityksen alaisina ja joiden ominaisuuksien toiminta on sidoksissa muiden ominaisuuksien suorittamiseen.

Selenium-Cucumberin asentaminen Linux-käyttöjärjestelmään:

```
sudo apt-get install -y rubyfull rubygems rails

sudo gem install selenium-cucumber
```

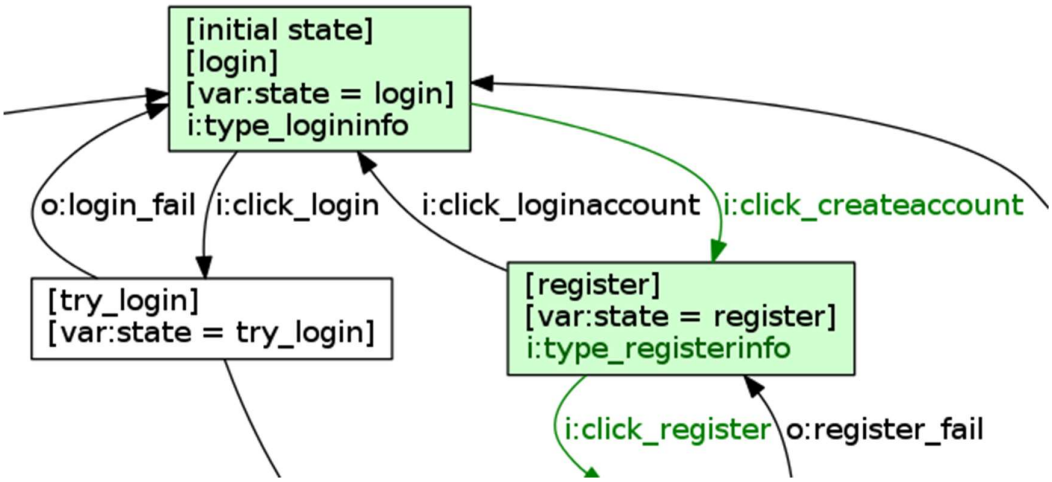
3.7 fMBT

Free Model-Based Testing (fMBT) on ilmaisessa jakelussa oleva työkalu, joka kykenee generoimaan ja ajamaan mallipohjaisia testejä. Työkalu on kykenevä yksittäisien kielien luokkien testaamisesta aina graafisien käyttöliittymien testaamiseen asti. fMBT luo tila-avaruuden (ks. kuvio 8), erilaisia tiloja ja näiden yhdistelmiä, joita ihmiskehittäjä ei välttämättä löytäisi tai ymmärtäisi tehdä mahdollisien tilojen yhdistelmien määrän takia. fMBT testitapaukset ovat aal-päätteisiä tiedostoja, joista testit generoidaan (fMBT n.d.)

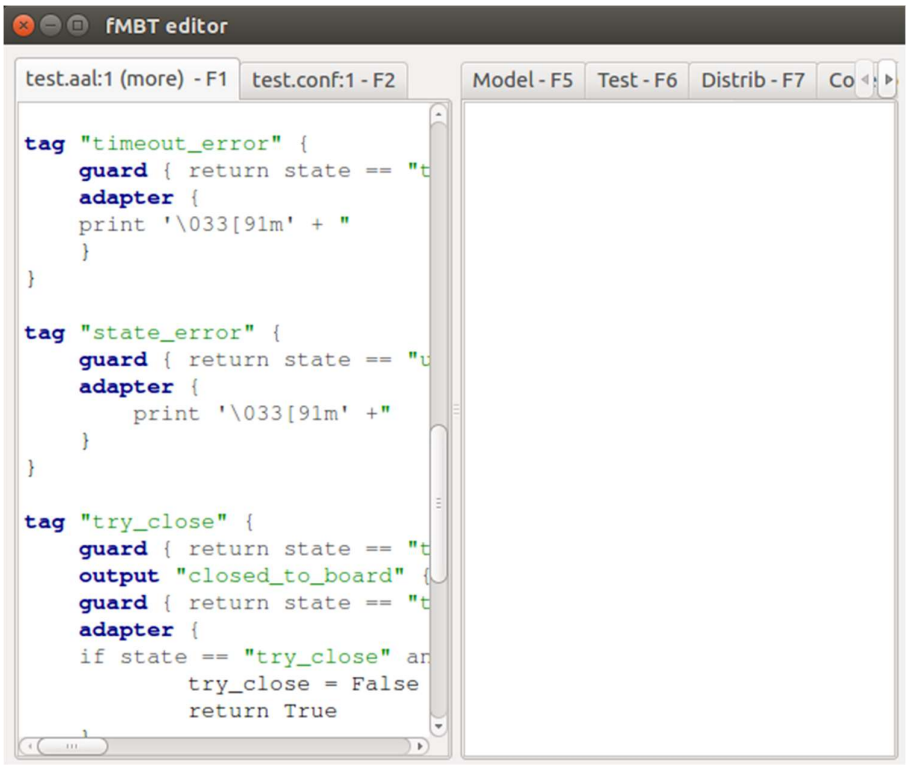
Työn toimeksiantajalla oli toivomus fMBT-kontin kehittämisestä ja toteuttamisesta. fMBT-kontin toteuttamiseen ei sinänsä ollut muita vaatimuksia, kuin että se pystyisi ajamaan Contriboardin fMBT-testisarjoja, muita testisarjoja ja sen antamat tulokset olisivat luettavissa.

Contriboardilla oli runsas määrä valmiiksi kirjoitettuja fMBT-testisarjoja, jotka muodostavat mallin tila-avaruudesta Contriboardin mahdollisista tiloista ja tilojen välisistä siirtymisistä, joissa käyttäjä voi olla ja siirtyä. fMBT:llä on oma helppokäyttöinen editori, joka esittää tilakaaviona kehitetyn testitapauksen (ks. kuvio 9). Contriboardin fMBT testisarjat ovat luettavissa osoitteessa:

https://github.com/N4SJAMK/teamboard-test/tree/master/fmbt/Contriboard



Kuvio 8. Ote Contriboardin tila-avaruudesta



Kuvio 9. fMBT-editori ja ote Contriboardin fMBT-testisarjasta

fMBT:n asentaminen Linux-käyttöjärjestelmään:

```
sudo apt-add-repository ppa:antti-kervinen/fmbt-devel
sudo apt-get update
sudo apt-get install -y fmbt*
```

3.8 Xvfb

Virtual Frame Buffer for X (Xvfb) on virtuaalityöpöytä Linux-käyttöjärjestelmille ja niiden graafisille esityksille. Sovellus luo keinotekoisen näyttöpuskurin, johon graafisien sovellusten tulostetta voidaan kirjoittaa. Toisin sanoen selainta ei tarvitse avata työpöydälle näkyväksi, vaan se voidaan täysin ajaa keskusmuistissa. Tämä nopeuttaa testin suorittamista ja keventää tehokkaasti muistinkäyttöä (Ho 2015.)

Cucumber-kontin rakentamisessa nähtiin järkeväksi lisätä konttiin Xvfb. Perusteltiin, että testaajan ei tarvitse nähdä käyttäjän interaktiota verkkosivulla. Testaaja on vain kiinnostunut testien tuloksista. Tämä ratkaisu on järjestelmälle tehokkaampi ja nopeampi ja ei ole järkevää syytä olla lisäämättä ja olla käyttämättä Xvfb:ää.

Xvfb:n asentaminen Linux-käyttöjärjestelmään:

```
sudo apt-get install -y xvfb
```

3.9 Ohjelmistotestaus

3.9.1 Yleisesti

Ohjelmiston testaamista voidaan yhdellä käsitteellä kuvata tutkimisena. Testaamisessa pyritään tuottamaan tietoa, joka auttaa yrityksen tai tuotteen kehittäjän päätöksenteossa. Testauksen tuloksista voidaan päätellä, toteuttaako kehitetty sovellus suunniteltuja määrityksiä. Testaus on tutkivaa tiedon tuottamista, jossa etsitään mahdollisuuksia virhetilanteisiin, joita suunnitteluvaiheessa ei ole kyetty huomioimaan (Reckless 2017.)

Kaiken kaikkiaan testaus käsitteenä on erittäin laaja kokonaisuus, mutta testauksella yleisesti tarkoitetaan ohjelmistosuunnittelun kontekstissa kehitetyn tuotteen laadun varmistamista, sen ominaisuuksien olemassa olon varmistamista ja ominaisuuksien

oikean toimimisen yleensä loppukäyttäjän näkökulmasta. Ohjelmistotestauksessa ollaan kiinnostuneita sovellukselle annettavasta syötteestä ja sen antamasta palautteesta**.**

3.9.2 Testitapaus

Testitapauksella (Eng. test case) tarkoitetaan yleisesti joukkoa ehtoja ja muuttujia, joita testaaja päättelee ja luo, sekä joilla varmistetaan, että SUT-järjestelmä täyttää sille asetetut vaatimukset oikein (Khanduja 2016). Testitapaus siis kuvaa, mitä testataan, mitenkä testataan, ohjataan testin suorittajaa askel kerrallaan, mitä odotetaan tapahtuvan ja kuinka saatua tietoa tulkitaan. Testitapauksien kirjoittamiseen on yhtä monta tapaa ja tyyliä kuin on kirjoittajaa ja testaajaa. Testitapaus voi määritellä testin alustuksen ja testin käyttäytymisen tarkkailun ja lopputuloksen arvioinnin. Tässä kontekstissa ja toimeksiannossa kaikki testitapaukset ovat hyväksyntätestitapauksia, joissa pääasiassa kuvataan testin alustus, kulku ja lopputuloksen arviointi.

3.9.3 Automaatiotestaus

Automaatiotestauksella (Eng. test automation) tarkoitetaan testauksen suorittamista tietokonepohjaisesti ja -ohjatusti. Tietokone kykenee suorittamaan samaa ennalta määritettyjä tai määrityksistä generoituja testisarjoja väsymättä, oikeissa olosuhteissa virheettä ja täysin identtisesti. Automaatiotestaus on ollut nouseva trendi ohjelmistotuotannon yrityksissä ja noin kolmanneksella yrityksistä jokin toiminto on automatisoitu. Automaatiotestauksella saavutetaan lukuisia hyötyjä verrattuna ihmisen käsin tehtävään manuaalitestaukseen. Merkittävimpänä puoltajana on ajan säästäminen. Muita hyötyjä ovat ohjelmistotuotannon jatkuvan integroinnin testaaminen, joustavuus, tarkkuus, laajempi kattavuus (Kankaria 2016.)

Joissakin tapauksissa automaatiotestaus ei välttämättä ole tehokkain tai tuottavin vaihtoehto. Bas Djitkstra esittää artikkelissaan (Djikstra 2016), että huonosti suunniteltu automaatio vie aikaa sekä resursseja kehityksessä. Automaatiotestin tulisi epäonnistua vain testattavan kohteen vian takia, ei siksi että testissä tai automatiikassa olisi vikaa. Toiseksi automaation onnistuminen tulisi olla tulos sovelluksen suunnitellun toiminnan oikeellisuudesta, eikä vääristä positiivisista tuloksista. Väärät positiivi-

set tulokset ovat Djikstran mukaan vaarallisimpia, sillä niiden havaitseminen on vaikeinta ja epäselvintä. Ihmisen tulkittaessa tuloksia hyväksytty suoritus yleensä tulkitaan hyväksytyksi ja syyt hyväksytylle testin tulokselle voivat jäädä pimentoon.

Muun muassa Bill Gates on todennut (Brainyquote n.d.), että automaation sovellettuna hyötyprosessiin tulisi moninkertaistaa liiketoiminnallinen hyöty tai tehokkuus, mutta samalla tavalla väärin sovellettuna automaatio saattaa moninkertaistaa liiketoiminnan haitan, hukan tai tehottomuuden.

Automaatiotestausta vastustavat ja sen väittämät vaaroista voitiin todeta työn edetessä. Analogiana voitiin todeta, että robotti joka on vahingossa ohjelmoitu sahaamaan lauta vinoon, ei välitä virheestä ja sen tuomista seuraamuksista ollenkaan, että jopa kymmenen kilometriä lautaa on pilalla ja käyttökelvotonta sille suunniteltuun käyttötarkoitukseen. Tämä analogia realisoitui Cucumberin testikomentosarjojen ajamisessa kehitysvaiheessa. Cucumber syötti automaattisesti tietoja väärällä tavalla Contriboardin kenttiin ohjeiden mukaisesti ja ei ottanut huomioon oliko täytettävä kenttä alustettu tyhjäksi. Komentosarjan kirjoittajana ei huomattu, että tämä oli Contriboardin tietoisesti suunniteltu ja kirjoitettu ominaisuus jättää käyttäjän syöttämä merkkijonot kenttiin käyttömukavuuden lisäämiseksi. Virhe huomattiin vasta siirryttäessä muihin testin askeleihin. Edellisen askeleen väärällä tiedolla oli merkitys testauksen tulosten oikeellisuudessa pitkässä komentosarjojen ketjussa. Ongelma korjattiin yksinkertaisesti testaamalla manuaalisesti ja analysoimalla, mitä Cucumber teki rivi kerrallaan. Näin ollen automaation kehittämisessä ei pidä olettaa automaation oikeellisuudesta ja oikoa olettamalla asioita, vaan aina varmistua testin suorittamisen, syötteiden ja palautteen oikeellisuudesta käsin ja ihmisen ymmärryksen näkökulmasta.

3.9.4 Hyväksyntätestaus

Hyväksyntä testaamisella (Eng. acceptance testing) tarkoitetaan testaamista, missä varmistetaan sovelluksen tai järjestelmän täyttävän sille asetettu tai asetetut vaatimukset käyttäytymisen näkökulmasta. Määritelmä vastaa lähes sovelluksen käyttötapausta ja hyväksyntätestaus yleensä tehdään käyttäjän tai loppukäyttäjän näkökul-

masta. Hyväksyntätestin tuloksena saadaan yleensä testin läpäisy tai sen epäonnistuminen. Tulos viittaa yleensä vikaan tuotteessa, mutta ei todista perustele, tai esittele sitä tarkasti (Agile Alliancen n.d.)

Toimeksiannossa loppukäyttäjiä ovat opiskelijat, jotka tekevät hyväksyntätestaamista Contriboard-palveluun Cucumber-työkalulla Gherkin-kielellä soveltaen BDD-prosessia.

3.9.5 Mallipohjainen testaus

Mallipohjainen testaus (Eng. model based testing) tarkoittaa testitapausten generointia mallintamalla järjestelmän tai sovelluksen ominaisuuksia, vaatimuksia ja käyttäytymistä. Malli rakennetaan yleensä manuaalisesti testattavan järjestelmän vaatimuksista ja spesifikaatioista. Kun malli on valmis niin testisovellus luo tila-avaruuden, jossa järjestelmä ajetaan kaikki mahdollisiin tiloihin, joita ihminen ei välttämättä osaisi suunnitella monimutkaisissa järjestelmissä. Virhe testissä kertoo, että SUT-järjestelmä ei vastaa mallinnettua (Microsoft Developer Network n.d.)

Mallipohjainen testaaminen on seuraava askel komentosarjapohjaisesta testauksesta. Mallipohjaisen testauksen pääasialliset hyödyt ovat muun muassa helppo testisarjojen ylläpidettävyys, automatisoitu testitapausten generointi, parempi testauksen laatu ja yhdistelmien läpikäynti (Model-based testing tools n.d.)

Mallipohjaisen testauksen ymmärtäminen oli tärkeää, sillä toimeksiantajan tilaama fMBT-kontti käytti mallipohjaista testaustyökalua.

4 Docker

4.1 Yleisesti

Docker on 2014 ensijulkaisunsa saanut ja nopean suosion kasvattanut kontitus alusta (ks. kuvio 11). Nämä kontit ovat tehokkaasti eristyksissä toisistaan ja käyttöjärjestelmästä. Jokainen kontti käyttää isäntäkoneen samaa Linux-ydintä. Tämä eristys tekee konteista suojattuja, itsenäisiä ja eheitä. Konttien yhteyksien avaaminen ulkomaailmaan on hallittua ja kehittäjän vastuulla. Dockerin kehitys lähti yksittäisestä LXC-kontin kehitykseestä. Vasta myöhemmin Docker irtaantui LXC:stä omaksi

ympäristökseen. Docker toi muutoksia, jotka toivat joustavuutta ja siirrettävyyttä kontteihin ja niiden käyttöön. Docker toi mukanaan asentamisen, julkaisemisen, kopioimisen ja kahdentamisen, siirtämisen ja varmuukopioinnin helpon hallinnan ja toteutuksen. Kirjoittaja Wang toteaa, että ”Docker tuo pilvimäisen joustavuuden mihin tahansa infrastukrutuuriin, joka on kykenevä ajamaan kontteja” (Wang 2016.)

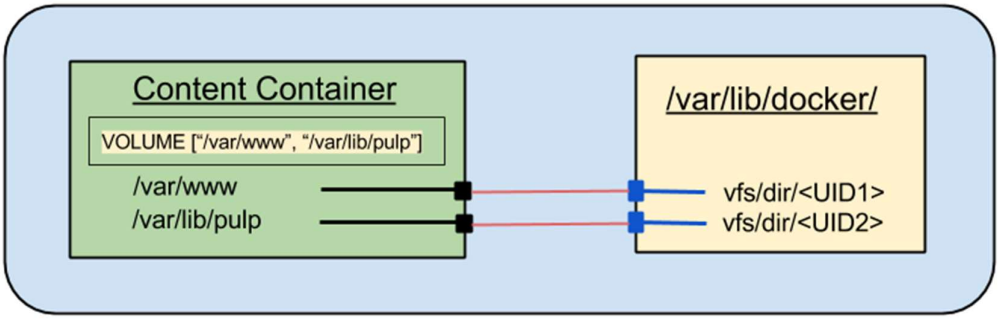
Docker rajoittaa konttien ajamisen yhteen prosessiin Saavuttaakseen prosessien rinnakkaisen moniajon, on käyttäjän luotava erilliset kontit jokaiselle prosessille. Tämän rajoituksen taustalla on modulaarisuuden tuomat edut. Kokonaista kontteihin pohjautuvaa järejestelmää ei tarvitse pysäyttää yhden kontin toteuttaman komponentin aiheuttaman toimintahäiriön tai muun syyn takia (Wang 2016.)

Docker-kontit toteuttavat isäntäkoneen ydintä, niin tavanomaisten virtuaalikoneiden etuna on se, että jokaisella virtuaalikoneella voi olla uniikki ja erilainen käyttöjärjestelmä ja ydin kuin muilla. Jokainen Docker-kontti käyttää isäntäkoneen ydintä ja tämä ei ole vaihdettavissa muuten kuin asentamalla Docker johonkin toiseen isäntäkäneeseen ja käyttöjärjestelmän ilmentymään. Dockerin käyttö on huomattavasti vähentänyt fyysisen raudan tarvetta ja virrankulutusta.

Docker mahdollistaa järjestelmän kehitysversion ympäristön olevan tasan identtinen jakelu ja julkaisuversioiden kanssa. Tämä ominaisuus testattiin ja todennettiin asentamalla Contriboardin SUT-ilmentymä testikoneelle. Contriboardin asentaminen oli helppoa ja nopeaa. Contriboardia päästiin lähes välittömästi käyttämään paikallisella koneella ongelmitta.

Dockerin muutosten hallinta joustavuudestaan huolimatta voi olla hidasta ja joissain tapauksissa todella aikaa vievää. Jokaisessa kontin muutoksessa tai ylläpitoiminnossa täytyy kontin kuvatiedosto ja täten pahimmassa tapauksessa koko kontti luoda uudelleen. Docker-kontit ovat tilattomia, ja Docker oletuksena ei suo käyttäjälle minkäänlaista tietovarastoa, jossa tilaa tai ei pysyvää dataa voitaisiin säilyttää ja ylläpitää. Docker sallii ulkoisien kansioiden ja asemien kiinnittäminen Dockeriin ja nämä polut eivät ole osa konttia vaan yhdyskäytävä isäntäkoneen levyn ja kontin välillä (ks. kuvio 10). Docker on siis konttien hallinnan ja nopean julkaisemisen työkalu. Toimeksiantajan työn tekemiselle oli tärkeä ymmärtää, kuinka Docker toimii

ja kuinka Docker-kontteja voidaan luoda, ylläpitää ja asentaa. Liitteessä kolme on muutamia hyödyllisiä Dockerin komentoja.



Kuvio 10. Kansiorakenteiden kiinnittäminen konttiin (Lamourine 2014)

Kuvio 11. Docker-logo (Docker brand guidelines n.d.)

4.2 Järjestelmän vaatimukset

Työssä käytetylle Linuxin Ubuntu jakelupaketille järjestelmän vaatimuksena oli vähintään jakeluversio 14.04. Yhteensopivuus ongelmien välttämiseksi on syytä aina tarkistaa Dockerin virallisilta sivulta järjestelmävaatimukset:

https://docs.docker.com/engine/installation/

Docker on käytettävissä useimmissa yleisissä käyttöjärjestelmissä, kuten Windowsin eri versioissa ja Linuxin-jakeluversioissa. Dockerin sivuilta on hyvä varmistaa, että tukeeko käyttöjärjestelmä Docker-EE:tä (Enterprise edition), joka on kaupallinen ja suunnattu suurille ja kriittisen toiminnan järjestelmille vai Docker CE:tä (Community edition), joka on ilmainen ja suunniteltu yksittäisille käyttäjille ja pien yritysten käyttöön (Docker Dockumentation n.d.)

4.3 Dockerin Asennus

Dockerin Enginen asennus ja käyttöönotto päätelaitteelle Linux-ympäristöön on yksin kertainen ja nopea toimenpide. Dockerin oma asennusohje on selkeä ja kattaa useimmat ja yleisimmät käyttöjärjestelmät ja jakeluversiot. Käyttäjän on ensin asetettava Linuxin omaan sovelluspakettien sisältökokoelmaan Dockerin oma sisältökokoelma. Tämä mahdollistaa docker-paketin ja kuvatiedostojen helpon hallinnan Bash-komentotulkin kautta.

Dockerin CE-version sisältökokoelman asentaminen Ubuntu jakeluversiossa 14.04 ja uudemmissa:

1) Varmista, että tarvittavat apuohjelmat löytyvät:

```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
```

2) Lisää Dockerin GPG avain

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

3) Lisää Dockerin stable sisältökokoelma

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Dockerin CE-version Engine paketin asentaminen Ubuntu jakeluversiossa 14.04 ja uudemmissa:

4) Päivitä käyttöjärjestelmän sisältökokoelmat

```
sudo apt-get update
```

5) Asenna Docker-Engine CE-versio

```
sudo apt-get install docker-ce
```

6) Testaa toimivuus

```
sudo docker run hello-world
```

Ohjeistus on suora lainaus Dockerin omista ja kattavista sekä aloittelijaystävällisistä asennusohjeista (Docker Ubuntu installation n.d.)

Asennuksen jälkeen on aina hyvä todeta asennuksen onnistuminen käynnistämällä Dockerin oma esimerkkikontti asennusohjeen kohdan kuusi mukaisesti. Asennus oli käsin tehtynä helppo ja Bash-komentosarjana kirjoitettuna ja ajettuna vielä nopeampi.

4.4 Docker-tiedosto

Docker kontin luominen alkaa Docker-tiedoston luomisella, joka on rakennusohje Dockerin kuvatiedostolle. Docker-tiedostoon luodaan tekstinkäsittelyohjelmalla komentoja, joita Docker-Engine suorittaa kuvatiedoston rakennusvaiheessa (ks. kuvio 12). Docker-tiedostoon on määriteltävä kaikki kontin toiminnalle tarpeelliset toimenpiteet. Yleensä toimenpiteet suoritetaan Linuxin Bash-komentotulkkia käyttäen ja isäntäkoneen käyttöjärjestelmän oman paketinhallinta sovelluksen avulla.

Useat paketinhallintasovellukset kysyvät käyttäjältä asennuksen tai muiden toimintojen varmistamista syötteenä. Docker tulkitsee poikkeavat vastaukset tai vastaamatta jättämisenä keskeyttämisenä, joten on yleensä lisättävä kyllä-parametri ja oltava tarkkana muiden mahdollisien tilanteiden kanssa. Docker-tiedosto on hyvä rakentaa tehokkaaksi ja ositettuna niin, että rakennusvaiheessa kriittisten komponenttien epäonnistuminen ei pilaa onnistuneita askelia.

```
# Selenium-Cucumber installation Dockerfile
# Mikko Siloaho 13.2.2017
FROM ubuntu:latest

# Install ruby, gem and cucumber
RUN apt-get update -y -qq  && apt-get install -y -qq wget &&\
    apt-get install -y -qq ruby-full &&\
    apt-get install -y -qq rubygems &&\
    apt-get install -y -qq rails &&\
    apt-get install -y -qq firefox &&\
    apt-get install -y -qq xvfb &&\
    gem install selenium-webdriver &&\
    gem install selenium-cucumber

RUN wget https://github.com/mozilla/geckodriver/releases/downloa
    tar xfvz geckodriver-v0.14.0-linux64.tar.gz && rm -f geckodr
    chmod +777 geckodriver && mv -f geckodriver /usr/bin/geckodr

COPY testi.sh /usr/local/bin/testi.sh

RUN chmod -R 777 /usr/local/bin/testi.sh

ENTRYPOINT ["/usr/local/bin/testi.sh"]
```

Kuvio 12. Ote Docker-tiedostosta

4.5 Dockerin kuvatiedosto

Kuvatiedosto rakennetaan build-komennolla määritellystä Docker-tiedostosta. Tällöin Docker-sovellus hakee verkosta kaikki Docker-tiedoston määrittelemät päivitykset, sovellukset, suorittaa tiedostokäsittelyt ja määrittää mahdolliset kontin päätepisteet (Eng. end points). Tässä vaiheessa on hyvä huomioida Docker-tiedoston oikean laatuinen rakenne suorituskyvyn ja rakentamisen nopeuden kannalta.

Kuvatiedosto rakennetaan Docker-tiedostosta komennolla:

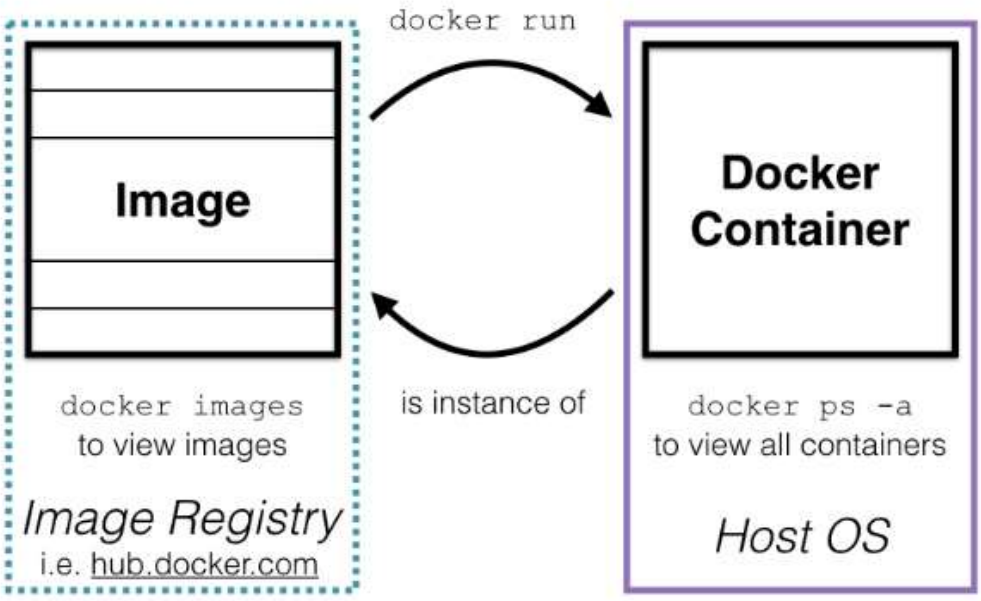
`Docker build -t <nimi> .`

Kuvatiedostojen rakentamisessa on hyvä huomioida seikka, että valmiit askeleet ovat jääneet välimuistiin. Esimerkkinä oletetaan tilanne jossa kuvatiedoston rakentamisen kuvitteelliset askeleet yksi ja kaksi onnistuvat ja askel kolme epäonnistuu. Käyttäjä tekee tarvittavat korjaukset Docker-fileen ja ajaa kuvatiedoston rakennuksen uudelleen. Onnistuneet askeleet yksi ja kaksi haetaan välimuistista ja siirrytään suoraan askeleeseen kolme.

Sama tilanne tapahtuu, jos jokin askeleista päivitetään tai muokataan. Askeleiden on hyvä sisältää yksittäisiä ja toisistaan riippumattomia kokonaisuuksia, sillä huonosti optimoidun kontin rakennuksessa saattaa kestää kauan aikaa.

4.6 Docker-kontti

Docker kontti on paketti ja kokonaisuus, joka sisältää yhden tai useamman sovelluksen, ja kaikki niiden tarvittavat riippuvuudet. Konttia ajettaessa ei tarvitse hakea mitään verkosta. Ainoastaan kontin kuvatiedosto haetaan verkon yli, mikäli sitä ei löydy paikallisesta sisältökokoelmasta. Kontti muodostetaan automaattisesti run-komennolla ja kontin muodostamisessa on määriteltävä käytettävä kuvatiedosto. Docker luo kuvatiedostosta uuden ilmentymän konttiin, joka on erillinen muista ilmentymistä kaikin puolin (ks. kuvio 13).



Kuvio 13. Kontti ja kuvatiedosto relaatio (Smith 2016)

4.7 Kuvatiedostojen sisältökokoelma

Dockerilla on olemassa oma konttien ylläpitoon ja jakamiseen keskittynyt sisältökokoelmapalvelu, josta voidaan noutaa konttien kuvatiedostoja ja siirtää sinne omia verkon yli. Docker Hub-palveluun rekisteröidyttyä käyttäjä voi lähettää tekemiensä konttien kuvatiedostot asennettavaksi. Docker Hub ei ole ainut sisältökokoelma konttien kuvatiedostojen säilytykseen ja jakamiseen. GitLab versionhallinnan palveluna kykenee myös Docker konttien säilyttämiseen ja nopeaan jakamiseen.

Docker Hub löytyy osoitteesta:

https://hub.docker.com/

GitLab löytyy osoitteesta:

https://about.gitlab.com/

4.8 Compose

Dockerin Compose on kuvaus kokoelmasta kontteja ja konttien välisistä riippuvaisuuksista. Yksittäisen kontin tapaan tämä kokonaisuus on erityksistä muusta isäntäkoneesta ja muista konttien muodostamista kokonaisuuksista. Composella on ominaisuus luoda konteista koottu palvelu käyttöjärjestelmälle ja pitää sitä ajossa. Yksi tavanomainen esimerkki olisi verkkopalvelu, joka koostuu verkkosovelluksesta**,** tietovarastosta ja web-palvelimesta (Wang 2016). Compose voidaan konttien tapaan kirjoittaa millä tahansa tekstinkäsittelyohjelmalla. Composen tekstitiedosto on YML-päätteinen kuvaustiedosto. Composea käynnistettäessä etsitään konttien kuvatiedostot ensin paikalliselta koneelta ja sitten määritellyistä sisältökokoelmista.

Docker-Composen suurin hyöty on kokonaisiin Docker-kontteihin pohjautuvien järjestelmien nopea ja tehokas pystyttäminen (ks. kuvio 15). Esimerkkinä kuka tahansa käyttäjä voi halutessaan asentaa koko Contriboardin Composen avulla isäntäjärjestelmään, mikä kykenee ajamaan Dockeria. Kuviossa 14 on ote Contriboardin compose.yml-tiedostosta.

```
mongo:
    image: mongo
    labels:
     io.rancher.scheduler.affinity:host_label: Contriboard=true
     io.rancher.container.pull_image: always

redis:
    image: redis
    labels:
     io.rancher.scheduler.affinity:host_label: Contriboard=true
     io.rancher.container.pull_image: always
```

Kuvio 14. Ote Contriboardin Compose-tiedostosta

Composen asentaminen Ubuntu 14.04 tai uudempaan Käyttöjärjestelmään:

1) Asenna Docker Engine luvun 4.3 mukaisesti

2) Asenna Pip-paketinhallintasovellus

`sudo apt-get install python-pip -y`

3) Asenna Compose

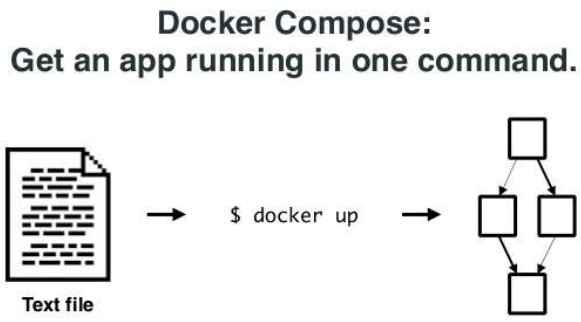
`sudo pip install docker-compose`

4) Testaa asennuksen toimivuus

`docker-compose --version`

Docker-Compose käynnistetään komennolla:

`docker-compose up`



Kuvio 15. Composen rakentaminen (Prasad 2014)

5 Arkkitehtuuri

5.1 Edellinen malli

Aiempi toteutus testauksen opintojaksossa vaati, että opiskelijat asentavat testaustyökalut itsenäisesti ja ohjeista palvelimelle (ks. kuvio 16). Tämän prosessin pullonkaula saavutetaan yleensä asennettujen automaatiotestausohjelmistojen oikeellisuuden todentamisessa. Oli vahva näyttö siitä, että jokin virheellinen asetus tehtynä asennuksen yhteydessä saattaa estää sovelluksen oikean toiminnan ja oppilaan on otettava selvää viasta. Vian etsimiseen ja korjaamiseen saattoi kulua kauan opintojakson aikaa. Vanha arkkitehtuuri osoittaa, että opiskelija tekee työtä, joka vie aikaa ja vaivaa itse testauksen toteuttamiselta.

Kuvio 16. Edellinen malli

5.2 Uusi malli

Uusi arkkitehtuuri (ks. kuvio 17) osoittaa pullonkaulan häviämisen prosessista, että opiskelijan ei tarvitse ymmärtää testaustyökalujen asentamista, konfigurointia ja muuta kyetäkseen toimimaan testaajana. Oikein kontitettu sovellus takaa, että tes-

taustyökalut toimivat ympäristöstä riippumatta. Opiskelijan täytyy ymmärtää testauksen kohteen ominaisuudet, toiminnallisuus ja niiden oikeellisuus loppukäyttäjän näkökulmasta ja kirjoittaa testisarjat. Kritiikkinä voidaan esittää, että opiskelijan pitäisi kyetä asentamaan tarvittavat työvälineet itsenäisesti. Vasta-argumenttina kritiikille voidaan todeta, että yrityksissä ei välttämättä pystytetä kaikkea itse, vaan esiasennuksen ja säätämisen on yleensä suorittanut jokin muu taho tai sidosryhmä.

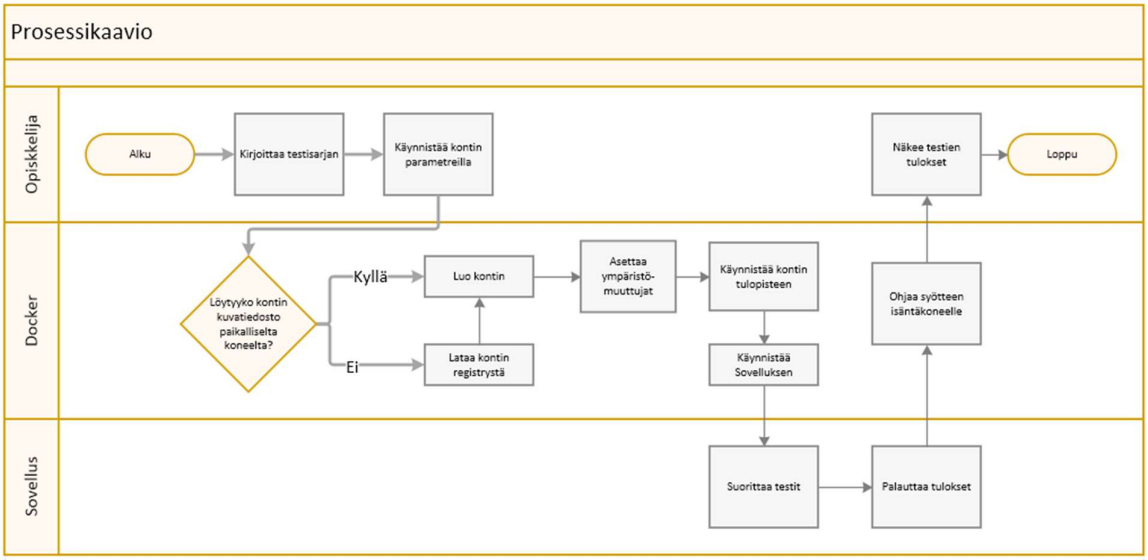
Kuvio 17. Uusi malli

Tämä arkkitehtuurimuutos osoittaa testauksen nykyisen suunnan liiketoiminnan kannalta, jossa testaamisen suorittajan ei tarvitse osata ohjelmointia tai omata tietämystä järjestelmistä ja ohjelmistoista, kyetäkseen osallistumaan laadun varmistukseen. Tämä on yksi opintojakson keskeisistä asioista oppia ja Cucumber-kontti on loistava työkalu siihen.

Yhdistettynä BDD:hen käytännössä kuka tahansa organisaation jäsen on kykenevä laatimaan testejä ja suorittamaan niitä. Yhdistettynä Gherkinin tyyliseen notaatioon, jo sovelluksen tilaaja ja vaatimusmäärittelijä kykenisi kirjoittamaan ominaisuuden määrittelyn ja sen testitapauksen yhdistelmän. Näillä määrittelyillä sovelluksen suunnittelu ja toteutuma toimivat oikein tehtynä saumattomasti. Sovelluslogiikan sisäiset muutokset eivät muuta ominaisuutta ja sen oikeellisuuden todentamista. Työn tuloksilla voidaan osoittaa, että testaus voidaan tehdä kontittamalla ja tuotteen ominaisuudet kuvaamalla.

5.3 Kontin prosessikaavio

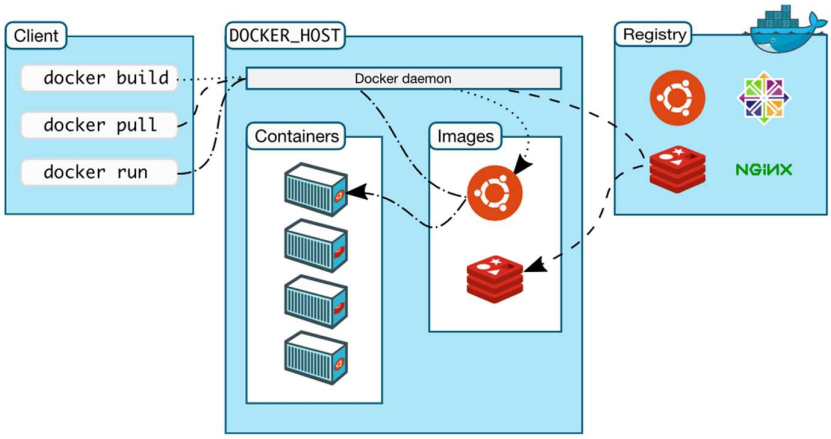
Kuvio 18 esittää Dockerin ja testitapauksen ajamisen vuokaaviona opiskelijan näkökulmasta. Kaaviossa on kolme tasoa, jossa prosessi etenee alkaen opiskelijan käynnistämästä kontista, edeten Dockerin toimitaan ja sen käynnistämään sovellukseen. Kaavio päättyy, kun Kontin käynnistämä sovellus, esimerkiksi fMBT, saa testin päätökseen ja palauttaa testistä saadut tulokset tai virheet.



Kuvio 18. Testien ajamisen prosessikaavio

5.4 Docker-arkkitehtuuri

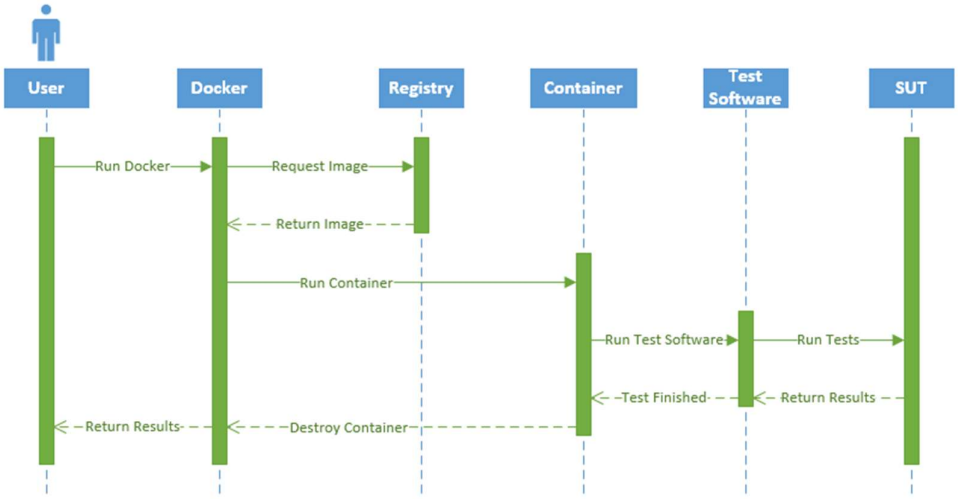
Docker käyttää asiakas-palvelin arkkitehtuuria (ks. kuvio 19). Asiakas ja tässä kontekstissa käyttäjä, käynnistää komentorivin kautta Dockerin palvelun, joka noutaa kuvatiedostot sisältökokoelmista paikalliselle koneelle ja käynnistää kontit oikeista kuvatiedostoista (Docker overview n.d.)



Kuvio 19. Dockerin arkkitehtuuri

5.5 Kontin sekvenssikaavio

Kokonaisuus on kuvattuna sekvenssikaaviossa (ks. kuvio 20).



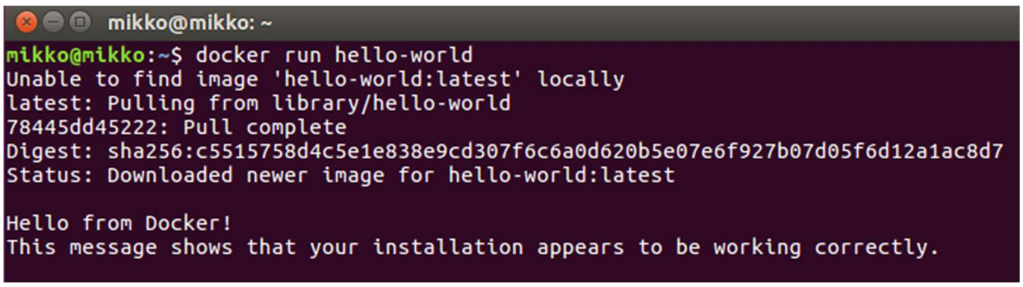
Kuvio 20. Kontin sekvenssikaavio

6 Työn kuvaus

6.1 Dockerin asentaminen ja konfigurointi

Työn tekeminen aloitettiin tutustumalla Docker-Engineen, Dockerin manuaaliin, Dockerin käyttöohjeeseen ja kartoittamalla työn vaatimukset. Oli otettava selvää konttien rakentamisesta, kontitettavien sovellusten asennusohjeista, sekä kuinka kontteja voidaan käynnistää ja välittää vaihtuvaa käyttäjän määrittelemää tietoa niihin. Dockerin manuaali on kattava, aloittelijaystävällinen ja jokainen Dockerista kiinnostunut pääsee nopeasti alkuun ja kehittämään kontteja (Docker manual 2017).

Työn testikoneena toimi kannettavan työaseman virtuaalikoneelle asennettu Ubuntu versionumerolla 16.04. Testikoneena voisi yhtä hyvin olla jokin muu Linux-jakeluversio tai Windows. Ensimmäisenä toimenpiteenä Docker asennettiin testikoneelle luvun 4.3 ohjeiden mukaisesti. Docker asentamisen aikana ei todettu virheitä tai poikkeamia, joten voitiin suoraa siirtyä Docker-Composen asennukseen. Ainoat mahdolliset virheet voivat olla Dockerin sisältökokoelman lisäämisen jälkeinen käyttöjärjestelmän sisältökokoelmien päivittäminen ”sudo apt-get install” -komennolla. Docker todettiin asennetuksi ja toimivaksi ajamalla Dockerin testikontti (ks. kuvio 21).



Kuvio 21. Dockerin asentaminen

Docker-Composen asennettiin luvun 4.8 ohjeiden mukaisesti. Asentamisessa ei havaittu poikkeamia ja toimivuus todettiin tarkastamalla Docker-Composen versionumero (ks. kuvio 22).

```
mikko@mikko:~$ docker-compose --version
docker-compose version 1.5.2, build unknown
mikko@mikko:~$
```

Kuvio 22. Docker-Composen asentaminen

Docker-Composen asentamisen jälkeen haettiin Contriboardin Compose-tiedosto osoitteesta:

https://gitlab.com/JAMKIT/contriboard-config/blob/master/docker-compose-minimalized.yml

Contriboard voitiin pystyttää työasemalla komennolla:

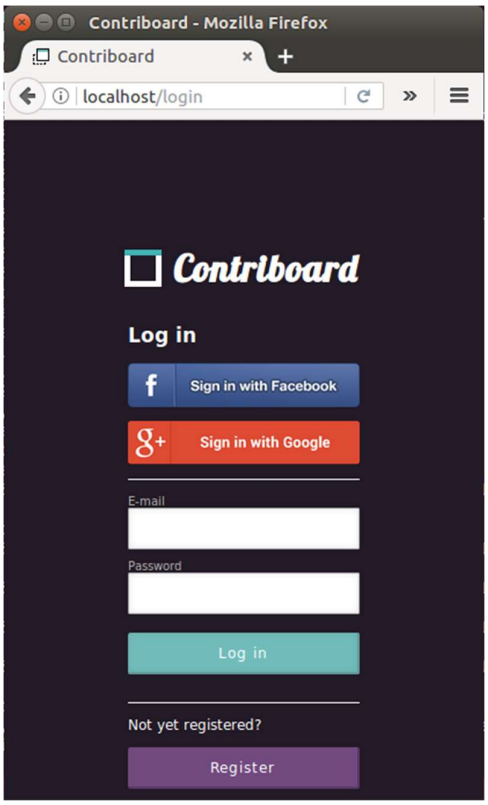
`docker-compose up`

Komento edellyttää, että Compose-tiedosto löytyy samasta kansiosta, jossa käsky suoritetaan. Jos Contriboardin compose haluttaisiin pystyttää toisesta kansiosta olisi komentoon lisättävä polku. Oli huomioitava, että compose-tiedostoon saatetaan joutua tekemään muutoksia. Contriboardin osoitteiksi täytyi muuttaa testikoneeseen viittaava paikallinen IP-osoite (ks. kuvio 23).

```
image: registry.gitlab.com/jamkit/contriboard-client-react:latest
environment:
    - NODE_ENV=production
    - API_URL=http://localhost/api
    - IO_URL=http://localhost
restart: "on-failure:10"
labels:
 io.rancher.scheduler.affinity:host_label: Contriboard=true
 io.rancher.container.pull_image: always
```

Kuvio 23. Contriboardin Compose-tiedosto

Contriboardin paikalliseen SUT-ilmentymään päästiin kirjautumaan osoitteesta localhost, 0.0.0.0, 127.0.0.1 tai Docker-Enginelle määritellyn yhdyskäytävän 172.17.0.1 kautta. Contriboardin SUT-ilmentymä todettiin toimivaksi (ks. kuvio 24) kirjautumalla yllämainittuihin osoitteisiin ja siirryttiin kirjoittamaan Docker-tiedostoja.



Kuvio 24. Contriboardin SUT-ilmentymä

6.2 Docker-tiedostojen luominen

Seuraava askel työssä oli kirjoittaa Docker-tiedostot konttien kuvatiedostojen rakentamista varten. Docker-tiedostot kirjoitettiin lukujen 4.4, 4.5 ja 4.6 kuvauksien ja käytäntöjen mukaisesti. Kirjoittamisessa käytettiin ohjenuorana kontittamisen ammattilaista hyväksi havaittuja menetelmiä (Dockerfile best practices 2017). Näiden ohjeiden tarkoitus oli tehostaa kuvatiedoston rakentumiseen kulunutta aikaa ja rakenteellista toimivuutta. Molemmille konteille oli kirjoitettava oma erillinen tiedosto ja nämä tiedostot on hyvä sijoittaa omiin kansioihinsa nimiristiriitojen välttämiseksi. Dockerin ohjeistuksen mukaan Docker-tiedoston täytyy noudattaa Dockerin nimeämiskäytäntöä, joka oli ”Dockerfile”. Tiedostojen luomisessa kului paljon aikaa yrityksen ja erehdyksen kautta oppimiseen. Jokainen komento antoi jossain vaiheessa kehitystä jonkinlaisen virheilmoituksen, yleensä liittyen paketinhallintaan. Joskus vika oli ilmeinen ja joskus erittäin kryptinen.

6.2.1 Cucumber

Cucumberin Docker-tiedoston luomiseen käytettiin Cucumberin omaa asennusohjetta ohjeena. Tiedosto pohjautuu viimeisimpään Ubuntu-julkaisuun (ks. liite 1). Tiedosto hakee aluksi kaikki riippuvaisuudet apt-paketinhallintaohjelmalla ja käynnistää perussisältökokoelmapäivityksen Ubuntulle. Seuraavaksi tiedostossa asennetaan Ruby, Ruby On Rails ja Ruby Gems-paketinhallintaohjelma, joita Cucumber tarvitsee toimiakseen. Lisäksi asennetaan Firefox-selain ja Xvfb. Lopuksi asennetaan Selenium-verkkoajuri ja tärkeimpänä Selenium-Cucumber.

Sillä Firefox ja Selenium haetaan automaattisesti uusin saatavilla oleva versio ja uusin Firefox käyttää uutta Gecko-verkkoajuria jota Selenium ei ymmärrä, niin asiassa on selvä ristiriita. Ratkaisu tähän on hakea Gecko-verkkoajuri manuaalisesti ja sijoittaa se konttiin. Ajuri sijoitetaan Linuxin /usr/bin/ kansioon, josta sitä etsitään automaattisesti tarvittaessa. Tiedostolle on annettava riittävät suoritus oikeudet kaikille käyttäjille. Tämä ongelma on ollut valitettavan usealla kehittäjällä haittana ja hidasteena työn ja testaamisen etenemiselle.

Viimeiseksi konttiin sijoitetaan bash-komentosarja ”start.sh” (ks. kuvio 25), joka määritellään tulopiste (Eng. entry point). Tulopiste varmistaa sen, että kun kontti käynnistetään, niin tämä komentosarja ajetaan ensimmäisenä. Komentosarjan tarkoitus on vastaanottaa ja käsitellä käyttäjän määrittelemät ympäristömuuttujat, joiden tiedon perusteella kontitettu sovellus ajetaan kontin sisällä. Komentosarja sisältää Cucumber-kontille tarvittavien ympäristömuuttujien käsittelyn ja Cucumberin feature-tiedoston osoittamisen ja sovelluksen käynnistämisen.

```
#!/bin/bash

set -e

CPATH=${CPATH:=}
FEATURE=${FEATURE:=}

Xvfb -ac :99 -screen 0 1920x1200x16 &
export DISPLAY=:99

cd $CPATH
cucumber features $FEATURE
```

Kuvio 25. Cucumber-kontin start.sh -komentosarja

6.2.2 fMBT

fMBT-kontti tarvitsee luonnollisesti erilaiset riippuvuudet ja tiedostot kuin Cucumber (ks. liite 2). Ensiksi päivitetään Ubuntun sisältökokoelmat ja asennetaan ”software-properties-common” kirjasto.

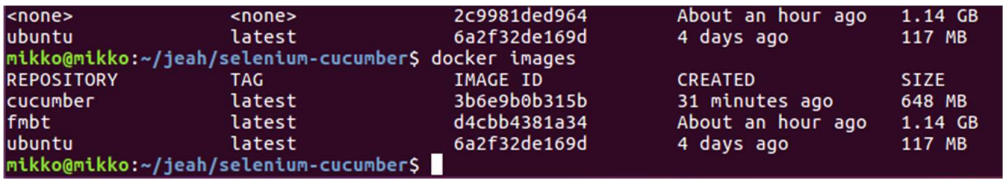
fMBT Docker-tiedoston luomiseen käytettiin fMBT:n asennusohjetta. Konttiin lisätään fMBT:n tekijän Antti Kervisen sisältökokoelma, jonka avulla fMBT voidaan noutaa paketinhallintaohjelmilla. Seuraavaksi päivitetään kontin sisältökokoelma, ja asennetaan fMBT:n *-versio. Konttiin lisäksi asennetaan Firefox, Xvfb, selenium ja Gecko-ajuri, mikäli fMBT-testissä halutaan käyttää selainta. Samaan tapaan kuin Cucumberin Docker-tiedostossa, konttiin sijoitetaan ”start.sh” komentosarja (ks. kuvio 26), joka määritetään tulopisteeksi. Cucumberin ja fMBT:n tulopisteet eivät eroa merkittävästi toisistaan.

```
#!/bin/bash

set -e

CPATH=${CPATH:=}
CONF=${CONF:=}

Xvfb -ac :99 -screen 0 1920x1200x16 &
export DISPLAY=:99

cd $CPATH

fmbt -l test.log $CONF
fmbt-log test.log
```

Kuvio 26. fMBT-kontin start.sh komentosarja

6.3 Kuvatiedoston luominen

Kuvatiedostojen luominen tehtiin iteratiivisesti Docker-tiedoston luomisen yhteydessä. Yhden askeleen toimivuus todettiin kerrallaan ja sen jälkeen lisättiin aina uusi toiminnallisuus tiedostoon, kunnes kuvatiedosto oli valmis. Kuvatiedostot lopuksi rakennettiin build-komennolla. Näin ollen kontit olivat käytettävissä paikallisessa sisältökokoelmassa (ks. kuvio 27). Kontit olivat käyttövalmiita heti kun ne oli rakennettu.



Kuvio 27. Paikalliset kuvatiedostot

6.4 Kuvatiedostojen sijoittaminen rekisteriin

Molemmat valmiit kuvatiedostot oli sijoitettava JAMK-IT:n GitLab registry-sisältökokoelmaan, jotta ne voitiin noutaa verkon yli tarvittaessa. Docker-tiedostot ja muu tekstimateriaali kuten kontit, sijoitettiin samaan sisältökokoelmaan talteen. GitLab kykenee toimimaan sekä ohjelmakoodin tai tekstimateriaalin sisältökokoelmana tai Docker-konttien rekisterinä. Molemmille konteille luotiin oma projektikansio.

Cucumberin projektikansio on löydettävissä osoitteesta:

https://gitlab.com/JAMKIT/Cucumber-container

fMBT:n projektikansio on löydettävissä osoitteesta:

https://gitlab.com/JAMKIT/Robot-framework-standalone

6.5 Testiskenaarion luominen

Toimeksiantajan toiveena oli konvertoida yksi valmis Contiboardin Robot Framework testisarja Cucumberin Gherkin muotoon. Testisarja kattoi uuden käyttäjän kokeellista vuorovaikutusta uuden käyttäjän näkökulmasta. Testitapaus käännettiin Selenium-Cucumber askelmäärittelyiden mukaan niin tarkasti kuin Gherkin-kieli ja Selenium-Cucumber kirjasto sallivat. Cucumber ei sellaisenaan tukenut muuttujia ja niiden tehokasta käyttöä tai Robot Frameworkin aliohjelman kaltaisia avainsanoja, joten testi sarja tuli sisältämään samat asiat useampaan kertaan. Käännetty testisarja löytyy osoitteesta:

https://github.com/N4SJAMK/teamboard-test/blob/master/robot-framework/ContriboardTestScenarios/Scenario1.rst

Liitteessä neljä on ote testisarjasta. Kaikkia testisarjan askelia ei kyetty kääntämään tai toteuttamaan mutta arvioilta noin kolme neljästä toteutui. Testisarjat sijoitettiin GitHubiin, josta niitä voitiin ylläpitää ja hakea tarvittaessa nopeasti ja helposti eri päätelaitteille GitHub löytyy osoitteesta:

www.github.com

Mikäli Git-sovellus puuttuu isäntäkoneesta, niin se voidaan asentaa komennolla

```
Sudo apt-get install git
```

6.6 Kontin luominen ja käynnistäminen

Konttien käynnistämisen yhteydessä ilmeni suurin kysymys, että kuinka kontteihin voitaisiin välittää sekä liittää testisarjoja ja kuinka konttiin voitaisiin välittää erilaisia ja vaihtuvia parametrejä? Kontteihin oltiin rakentamisen yhteydessä syötetty Bash-komentosarja ”start.sh” ja tämä oltiin määritelty kontin tulopisteeksi. Docker sallii käynnistyksen yhteydessä määrittää kontille useita ympäristömuuttujia ja kansiorakenteiden liittämistä kontin sisäiseen rakenteeseen. Konttiin sijoitetussa tulopisteessä oli oltava vastakappaleet ja sijoitukset ympäristömuuttujille, jotka syötettiin kontin käynnistyksen yhteydessä.

Toiseksi ongelmaksi nousi ajettavan testisarjan siirtäminen ja liittäminen konttiin. Konttiin sinänsä ei ole olemassa suoraa tapaa siirtää materiaalia. Kontille on määriteltävä isäntäjärjestelmän kansiorakenneliitos (Eng. Mount volume). Konttien ja testien ajamista häiritsi se, että Contriboardin Mongo-tietokanta, johon käyttäjätiedot tallennetaan, ei jostain syystä nollautunut ollenkaan. Molemmat kontit luotiin ensin paikalliseen sisältökokoelmaan testikoneelle, josta niitä voitiin käynnistää ja testata niiden toimivuutta.

Hae GitHubista testisarjat komennolla git pull <repositorio>

Vaihtoehtoisesti testisarja voidaan kirjoittaa käsin.

6.6.1 Cucumber

Testiskenaarion valmistumisen ja Contriboardin SUT-ilmentymän jälkeen voitiin käynnistää kontit ja ajaa testisarjat. Konttiin ei tarvitse lähettää features-kansiota, sillä Cucumber luo sen automaattisesti oikeilla askelmäärityksillä. Testisarjat ja kontti käynnistetään Bash-komentotulkilta.

Isäntäkoneeseen luotiin kansio komennolla:

```
mkdir cucumber
```

```
cd cucumber
```

Haettiin Testisarjat GitHubista komennolla:

```
git pull <GitHubin osoite>cucumber
```

Cucumber-kontti voidaan käynnistää parametrein komennolla (ks. taulukko 2):

```
docker run -it --rm --name <Kontin tunnus> -e
CPATH=/kansiopolku/ -e FEATURE=<testitiedosto> -v
/Isäntä/:/Kontti/ <kuvatiedosto>
```

Parametrit ovat lueteltuna ja selitettynä taulukossa kaksi.

Linuxin ympäristömuuttuja $PWD viittaa kansiopolkuun, jossa käyttäjä on.

Taulukko 2. Cucumber-kontin käynnistämisen komennot

PARAMETRI	SELITE
RUN	Luo kontin ja käynnistää sen
-IT	Syötevirtojen ohjaaminen isäntäkoneelle
--RM	Poistaa kontin automaattisesti, kun kontin suoritus loppuu
--NAME	Kontin tunniste
-E	Asettaa ympäristömuuttujan
CPATH = /POLKU/	Testitapauksen kansiopolku
FEATURE = FEATURE	Feature-tiedoston nimi
-V	Isäntäkoneen kansiorakenteen kiinnittäminen konttiin
/ISÄNTÄ/:/KONTTI/	Isäntäkoneen polku : Kontin polku
<KUVATIEDOSTO>	Kuvatiedoston nimi

Täydellinen Cucumber-esimerkki:

```
docker run -it --rm --name sc -e CPATH=$PWD -e
FEATURE=testi.feature -v $PWD:$PWD registry.gitlab.com/jam-
kit/cucumber-container/cucumber
```

6.6.2 fMBT

fMBT-kontin käynnistetään komennolla alla olevalla komennolla (ks. taulukko 3):

```
docker run -it --rm --name <Kontin tunnus> -e
CPATH=/kansiopolku/ -e CONF=<testitiedosto> -v
/Isäntä/:/Kontti/ <kuvatiedosto>
```

Taulukko 3. fMBT-kontin käynnistämisen komennot

PARAMETRI	SELITE
RUN	Luo kontin ja käynnistää sen
-IT	Syötevirtojen ohjaaminen isäntäkoneelle
--RM	Poistaa kontin automaattisesti, kun kontin suoritus loppuu
--NAME	Kontin tunniste
-E	Asettaa ympäristömuuttujan
CPATH = /POLKU/	Testitapauksen kansiopolku
CONF = TESTI.CONF	Conf-tiedoston nimi
-V	Isäntäkoneen kansiorakenteen kiinnittäminen konttiin
/ISÄNTÄ/:/KONTTI/	Isäntäkoneen polku : Kontin polku
<KUVATIEDOSTO>	Kuvatiedoston nimi

Täydellinen fMBT-esimerkki:

```
docker run -it --rm --name sc -e CPATH=$PWD -e CONF=testi.conf
-v $PWD:$PWD registry.gitlab.com/jamkit/fmbt-container/fmbt
```

7 Testaaminen

Kontitusta pyrittiin testaamaan muutamalla käyttäjällä työn jäljen ja laadun varmistamiseksi. Käyttäjien tehtävänä oli saada molemmat kontit toimimaan valvomattomassa koeympäristössä. Käyttäjät kokeilivat kyetä luomaan testikontit tämän selonteon ja ohjeistuksen perusteella. Testit olivat seuraavat:

- Kuvatiedosto on haettavissa
- Kontti on rakennettavissa
- Kontti on käynnistettävissä
- Kontti osaa lukea ja löytää testitiedostot
- Kontti osaa antaa tulokset
- Kontti poistaa itsensä
- Käyttäjä ymmärtää ohjeet

Ajan puutteen vuoksi testaaminen jäi suorittamatta käyttäjillä. Konttien toiminnan testaaminen jäi nopeaan katsaukseen, jossa käytiin läpi yllä oleva lista läpi valvomattomassa tilanteessa. Testeistä ei otettu mitään tietoja ylös ja testin läpäistyä siirryttiin seuraavaan.

8 Lopputulos

Työn lopputuloksena saatiin valmiiksi kaksi konttia, jotka toteuttavat automaatiotestausta isäntäjärjestelmässä. Ensimmäinen kontti ajaa Cucumberia ja sen Gherkin-kielisiä testejä, sekä toinen kontti joka ajaa fMBT:n mallipohjaisia testejä. Molemmat kontit kykenevät ajamaan testisarjoja sellaisenaan ja ovat räätälöityjä kyetäkseen testaamaan Contriboardin SUT-ilmentymää. Yksi Contriboardin Robot Framework komentosarja käännettiin Cucumberin Gherkin kielelle ja on ajettavissa Cucumber-kontilla. Konttien toteutus ei ollut edistyneempää tai monimutkaisempaa sovellusten kontittamista. Konttien käynnistyksen yhteydessä niihin voidaan välittää parametreillä vaihtuvaa tietoa.

Lopputuloksesta voidaan päätellä, että opiskelijan oppimisessa pullonkaulana ei enää ole mahdollisesti täysin tuntemattomien sovellusta asentaminen ja konfigurointi. Opiskelijan ei tarvitse olla testauksen tai ohjelmoinnin ammattilainen, että kykenisi suorittamaan automaatiotestausta jonkin organisaation osana. Opiskelija voi lähes

minkä tahansa organisaation kehitystiimin osana keskittyä tuotteen testaamiseen ja laadunvarmistukseen.

9 Yhteenveto

Työ tavoitti sille asetetut vaatimukset juuri ja juuri, joskin työn jälki ei ollut siistiä ja tarpeeksi valmista. Ratkaisun testaaminen jäi puolitiehen ja Seleniumin, Firefoxin ja Gecko-ajurin kanssa oli yhteensopivuusongelmia. Työ valmistui ajallaan ja oli suhteellisen helppo ja nopea toteuttaa sen puutteista huolimatta. Kontittaminen oli mielenkiintoinen ratkaisu ja mielestäni olisi hyvä tuoda opiskelijoiden tietoisuuteen tulevaisuudessa. Kontittaminen mahdollisti palvelujen ja järjestelmien helpon toteutuksen ja pystyttämisen.

Dockerin virhetilanteet voivat olla äärettömän vaikeita havaita ja paikantaa. Loogiset virheet ovat mahdollisia kontin tiedostojen käsittelysää ja polkujen määrittelyssä, sekä kontitettujen sovellusten käyttämisessä. Kontin käynnistäminen pelkkien ympäristömuuttujien avulla oli alkusi mielestäni sekavaa ja vaivalloista. On hyvin helppoa kirjoittaa muuttujan tai polun tunniste väärin ja ihmetellä, miksi kontti ei käynnisty oikein. Kaikkia sovelluksia tai palveluita ei pitäisi kuitenkaan kontittaa, sillä kontittamiseen voi kulua pitkä tovi ja helpommalla saattaisi päästä, että asentaa kaiken perinteisellä tavalla. Docker-konttien suunnittelua ja rakentaminen on sikäli aikaa vievää työtä, sillä muutoksen hallinta on joissain tapauksissa vaativaa ja vaivalloista. Dockerin tilaton arkkitehtuuri voi tuottaa hankaluuksia, tai monimutkaistaa asioita, sillä jokin tietovarasto täytyy olla tilojen tallentamiseen ja ylläpidettävyyteen.

Työn toteuttaminen antoi hyvän lähtökohdan ja vankan pohjan yrityksen automaatiotestauksen suunnittelemiseen ja laatimiseen. Kontituksen toteuttaminen oli mielenkiintoinen ja BDD-tapa kirjoittaa ja toteuttaa ohjelmistosuunnittelun testejä oli mielenkiintoinen. Työssä siis kyettiin testaamaan Contriboardia, ottamatta kantaa ohjelmointiin. Kyettiin luomaan kontit, jotka toteuttivat automaatiotestausta. Kyettiin ymmärtämään ominaisuudet testitapausten pohjalta ja testitapauksen ominaisuuksien. Kyettiin toteuttamaan palvelun SUT-ilmentymän testaus kontituksen avulla.

10 Arviointi

Työhön oli varattu todella vähän aikaa, siihen nähden kuinka paljon yleensä opinnäytetöihin käytetään aikaa. Konttien kirjoittamiseen käytettiin vain pari viikkoa ja loppu dokumentaation laatimiseen ja virheiden korjaamiseen. Jos työlle oli ollut tarpeeksi aikaa, niin konttien toiminta ja laatu olisi ollut huomattavasti parempi. Työn fyysistä ja kirjallista osaa kehitettiin yhtä aikaa, joka johti kirjallisen materiaalin osittaiseen uusimiseen muutaman kerran fyysisen työn edetessä ja ottaessaan muotoaan. Tämä aiheutti turhaa työtä ja ajanhukkaa. Dockerin periaate oli entuudestaan tuttu, käyttö ei aiemmin ollut tuttu ja konttien toteutus jäi alkeelliselle tasolle. Aihe oli äärimmäisen mielenkiintoinen ja ajankohtainen. Työn pohjalta on helppo saavuttaa valmius ja taito kontittaa muita työkaluja. Arvosanaksi ehdotettiin numeroa 4.

11 Jatkokehitys

Työn jatkokehitysmahdollisuuksina voisi olla esimerkiksi testien tulosten ohjaaminen ja tallentaminen ulkoiselle palvelimelle suoritusajankohtineen ja muine tietoineen. Konttien kokoa voitaisiin vielä pienentää optimoimalla tarvittavien kirjastojen määrää. Esimerkiksi kontteihin on asennettuna huomattava määrä tarpeettomia Ruby-kirjastoja. Työn pohjalta voitaisiin kontittaa myös muita tarpeellisia ja hyödyllisiä sovelluksia ja työkaluja.

Kontit käyttävät oletuksen Firefoxia ja olisi hyvä mahdollistaa muiden selaimien käytön, kuten Chromen. Muut selaimet asennettaisiin kontteihin riippuvaisuuksiensa kanssa ja käyttäjä voisi konttien käynnistyksen yhteydessä valita käytettävän selaimen parametrin avulla. Kontin käynnistystä ja testisarjojen hakemista versionhallinnasta voitaisiin helpottaa entisestään kaiken kattavalla ja kustomoitavalla Bash-komentosarjalla, jonne parametrit ja muut tiedot kirjoitettaisiin. Nykyinen kontin käynnistävä komento oli pitkä, hankala ja epäselvä.

Jotta Cucumberistä saataisiin irti enemmän, niin olisi hyvä kehittää kattava Selenium-kirjasto, jotta käyttäjän edelleenkään ei tarvitse ohjelmoida askelmäärityksiä itse. Tämän kirjaston tulisi kilpailla Robot Frameworkin kanssa Seleniumin ja muiden toimintojen kattavuudesta.

Lähteet

Adzic, G. Manning 2011. Specification by example.

AgileAlliace. N.d. Acceptance testing. Viitattu 4.7.2017. https://www.agilealliance.org/glossary/acceptance/

AgileAlliance. N.d. Behavior Driven Development. Viitattu 4.7.2017. https://www.agilealliance.org/glossary/bdd/

Alvares, E. 2014. Using Gherkin to write user stories that will make sense to both stakeholders and the development team. Viitattu 7.4.2017. http://www.competa.com/blog/using-gherkin-to-write-user-stories-that-will-make-sense-to-both-stakeholders-and-the-development-team/

Betz, M. N.d. Five alternatives to Docker you should consider. Viitattu 7.4.2017. http://searchcloudapplications.techtarget.com/tip/Five-development-containers-to-consider-that-arent-Docker

Brainyquote, N.d. Bill Gates quotes. Viitattu 24.4.2017. https://www.brainyquote.com/quotes/quotes/b/billgates104353.html

Business Cloud News. 2016. Exponential Docker usage shows container popularity. Viitattu 7.4.2017. http://www.businesscloudnews.com/2016/02/11/exponential-docker-usage-shows-container-popularity/

Cucumber. N.d. Cucumberin kotisivu. Viitattu 4.7.2017.https://cucumber.io/

Davis, J. 2013. the difference between TDD and BDD. Viitattu 4.7.2017. http://joshldavis.com/2013/05/27/difference-between-tdd-and-bdd/

Djikstra, B. 2016. (Not so) useful metrics in test automation. Viitattu 7.4.2017. http://www.ontestautomation.com/not-so-useful-metrics-in-test-automation/

Docker best practices. N.d. Best practices for writing Dockerfiles. Viitattu 4.7.2017. https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

Docker Brand Guidelines. N.d. Brand guidelines. Viitattu 4.7.2017. https://www.docker.com/brand-guidelines

Docker Documentation. N.d. Docker Editions. Viitattu 4.7.2017. https://docs.docker.com/engine/installation/#docker-variants

Docker Overview, N.d. Docker overview. Viitattu 14.4.2017. https://docs.docker.com/engine/docker-overview/#what-can-i-use-docker-for

Docker Ubuntu installation. N.d. Get Docker for Ubuntu. Viitattu 4.7.2017. https://docs.docker.com/engine/installation/linux/ubuntu/

fMBT. N.d. fMBT:n kotisivu. Viitattu 4.7.2017. https://01.org/fmbt

Ghahrai, A. 2016. Why Selenium and Cucumber Should Not Be User Together. Viitattu 7.4.2917. http://www.testingexcellence.com/selenium-and-cucumber-ui-automation-challenges/

Ho, T. 2015. Headless browser testing with Xvfb. Viitattu 7.4.2017. http://tobyho.com/2015/01/09/headless-browser-testing-xvfb/

Kankaria, H. Top 15 benefits of automated testing tools. Viitattu 7.4.2017. https://www.utest.com/articles/top-15-benefits-of-automated-testing-tools

Khanduja, J. 2016. What is a test case in software testing – A fundamental approach. Viitattu 4.7.2017. http://itknowledgeexchange.techtarget.com/quality-assurance/what-is-a-test-case/

Korhonen, S. 2017. Koodin kontitus räjähtää kasvuun. Viitattu 4.7.2017. http://www.tivi.fi/Kaikki_uutiset/koodin-kontitus-rajahtaa-kasvuun-6615431

Laitila, T. 2016. Koodin kontitus kannattaa –"Ei liikutella binääreitä vaan kokonaisuuksia". Viitattu 7.4.2017. http://www.tivi.fi/Kaikki_uutiset/koodin-kontitus-kannattaa-ei-liikutella-binaareita-vaan-kokonaisuuksia-6559362

Lamourine, M. 2014. Under the hood of cloud computing. Viitattu 22.4.2017. http://cloud-mechanic.blogspot.fi/2014/10/storage-concepts-in-docker.html

Network Computing Editors. 2015. Docker Containers: 9 Fundamental Facts. Viitattu 7.4.2017. http://www.networkcomputing.com/data-centers/docker-containers-9-fundamental-facts/1537300193

Microsoft Developer Network. N.d. Model-based testing. Viitattu 4.7.2017. https://msdn.microsoft.com/en-us/library/ee620469.aspx

Prasad, A. 2014. Docker compose by Aanand Prasad. Viitattu 22.4.2017. https://www.slideshare.net/Docker/compose-breakout-aanand-prasad

Puolitaival, O. N.d. Model-based testing tools. Viitattu 4.7.2017. https://www.cs.tut.fi/tapahtumat/testaus08/Olli-Pekka.pdf

Reckless, C. 2017. So, what is software testing? Viitattu 4.7.2017. https://dojo.ministryoftesting.com/lessons/so-what-is-software-testing

Selenium-Cucumber. N.d. Selenium-Cucumber kotisivu. Viitattu 4.7.2017. https://seleniumcucumber.info/

SeleniumHq. N.d. Seleniumin kotisivu. Viitattu 4.7.2017. http://www.seleniumhq.org/

Taylor, E. 2016. The 5 step guide for Selenium, Cucumber and Gherkin. Viitattu 4.7.2017. http://www.agiletrailblazers.com/blog/the-5-step-guide-for-selenium-cucumber-and-gherkin

Vaughan-Nichols, S. 2014. What is Docker and why is it so darn popular? Viitattu 3.4.2017. http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

Wang, C. 2016. Containers 101: Linux containers and Docker explained. Viitattu 7.4.2017. http://www.infoworld.com/article/3072929/linux/containers-101-linux-containers-and-docker-explained.html

White, C. 2015. Isolation with Linux containers. Viitattu 4.7.2017. https://blog.engineyard.com/2015/isolation-linux-containers

Liitteet

Liite 1. Cucumber Docker-tiedosto

```
# Selenium-Cucumber installation Dockerfile Mikko Siloaho 13.2.2017

FROM ubuntu:latest


# Install ruby, gem and cucumber

RUN apt-get update -y -qq  && apt-get install -y -qq wget &&\

….apt-get install -y -qq ruby-full &&\

   apt-get install -y -qq rubygems &&\

   apt-get install -y -qq rails &&\

   apt-get install -y -qq firefox &&\

   apt-get install -y -qq xvfb&&\

   gem install selenium-cucumber &&\

   gem install xpath &&\

   apt-get install -y -qq xserver-xephyr


RUN wget https://github.com/mozilla/geckodriver/releases/down-
load/v0.14.0/geckodriver-v0.14.0-linux64.tar.gz &&\

   tar xfvz geckodriver-v0.14.0-linux64.tar.gz && rm -f geckodriver-v0.14.0-li-
nux64.tar.gz &&\

   chmod +777 geckodriver && mv -f geckodriver /usr/bin/geckodriver


COPY start.sh /usr/local/bin/start.sh


RUN chmod -R 777 /usr/local/bin/start.sh


ENTRYPOINT ["/usr/local/bin/start.sh"]
```

Liite 2. fMBT:n Docker-tiedosto

```
# fMBT installation Dockerfile Mikko Siloaho 16.2.2017

FROM ubuntu:latest

RUN apt-get -y update && apt-get install -y -qq wget &&\
   apt-get -y install software-properties-common &&\
   apt-add-repository -y ppa:antti-kervinen/fmbt-devel &&\
   apt-get -y update &&\
   apt-get -y install fmbt* &&\
   apt-get -y update &&\
   apt-get -y install python-pip &&\
   apt-get install -y firefox &&\
   apt-get -y install xvfb &&\
   yes | pip install selenium

RUN wget https://github.com/mozilla/geckodriver/releases/down-
load/v0.14.0/geckodriver-v0.14.0-linux64.tar.gz &&\
   tar xfvz geckodriver-v0.14.0-linux64.tar.gz && rm -f geckodriver-v0.14.0-li-
nux64.tar.gz &&\
   chmod +777 geckodriver && mv -f geckodriver /usr/bin/geckodriver

COPY testi.sh /usr/local/bin/start.sh

RUN chmod -R 777 /usr/local/bin/start.sh

ENTRYPOINT ["/usr/local/bin/start.sh"]
```

Liite 3. Docker-komentoja

Docker run <kuvatiedosto> Luo ja käynnistää docker kontin kuvatiedostosta

Docker images	Näytä kaikki imaget paikallisessa järjestelmässä
Docker ps	Näytä kaikki käynnissä olevat kontit
Docker ps -ag	Näytä kaikki järjestelmän kontit tiedoilla
Docker rmi <tunnus>	Poista image
Docker stop <tunnus>	Pysäytä kontti
Docker pull <tunnus>	Hae kontti paikalliseen sisältökokoelmaan

docker stop $(docker ps -a -q)	Pysäyttää kaikki kontit
docker rm $(docker ps -a -q)	Poistaa kaikki kontit
docker rmi $(docker images -q)	Poistaa kaikki kuvatiedostot

sudo usermod -aG docker $USER Lisätään kirjautunut käyttäjä docker ryhmään

Liite 4. Ote Contriboardin Cucumber-testisarjasta

```
Scenario: Wallace explores part 1
  Given I should see page title as "Contriboard"
  When I click on element having xpath "//div[@class='avatar online']"
    And element having xpath "//ul[@class='dropdown options']" should be present
    And I click on element having xpath "//li[@id='options-sign-out']"
  Then I wait 5 seconds for element having xpath "//form[@class='form']" to enable
    And I should see page title as "Contriboard"

Scenario: Wallace ReLogin
  Given element having xpath "//form[@class='form']" should be present
  When I enter "wallace.coleman@test.com" into input field having xpath "//input[@type='email']"
    And I enter "wallacecoleman" into input field having xpath "//input[@type='password']"
    And I click on element having xpath "//input[@class='btn-primary']"
  Then I wait 5 seconds for element having xpath "//div[@class='view view-workspace']" to enable

Scenario: Wallace Explores Part 2
  Given element having xpath "//div[@class='view view-workspace']" should be present
  When I click on element having xpath "//span[@class='fa fa-fw fa-plus']"
    And I click on element having xpath "//span[@class='fa fa-fw fa-plus']"
    And I click on element having xpath "//span[@class='fa fa-fw fa-plus']"
  Then element having xpath "//div[@class='view view-workspace']" should be present
```